ESCUELA TÉCNICA SUPERIOR DE INGENIEROS

INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Aplicación de Asistente Virtual como Interfaz de Acceso a Plataformas de Ciudades Inteligentes

(On the use of Virtual Assistant as the Access Interface to Smart Cities' Platforms)

Para acceder al Titulo de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Raúl Cabria González

Octubre - 2020

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Raúl Cabria González
Director del TFG: Jorge Lanza Calderón
Título: "Aplicación de Asistente Virtual como Interfaz de Acceso a Plataformas de Ciudades Inteligentes"
Title: "On the use of Virtual Assistant as the Access Interface to Smart Cities' Platforms"

Presentado a examen el día: 30 de octubre de 2020

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Laura Bravo Sánchez
Secretario (Apellidos, Nombre): Jorge Lanza Calderón
Vocal (Apellidos, Nombre): Luis Sánchez González

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

Tras largos meses de aprendizaje y esfuerzo concluye este trabajo, pero no antes sin agradecer su apoyo, cariño y ayuda a todos aquellos que he sentido conmigo durante su realización, y no sólo eso, sino también durante los años que he dedicado al estudio de esta titulación.

En primer lugar, quiero agradecer a Jorge Lanza, tutor de este trabajo, su implicación en él y la ayuda que me ha ofrecido en todo momento.

En segundo lugar, agradecer a mi familia el apoyo y la confianza que ha tenido en mí desde el primer momento, aún cuando el transcurso del camino no parecía nada sencillo. Agradecer también a Natalia, por ser quien está conmigo siempre, quien me sustenta cuando yo no puedo. Los últimos meses de este trabajo hubiesen sido mucho más difíciles sin ella.

También quiero agradecer a todo el personal de la escuela que ha formado parte de mi desarrollo profesional y personal en los últimos años, tanto profesores como estudiantes, haciendo una mención especial a Jorge Calleja, compañero y amigo.

Resumen

Los asistentes virtuales se nos ofrecen como una de las tecnologías revolucionarias que existen en el presente y la base para la interacción humano – máquina en los sistemas del futuro. Esta tecnología permite el desarrollo de sistemas más amigables y cercanos al usuario, como podemos encontrar en los hogares inteligentes, permitiendo la gestión de casi cualquier dispositivo electrónico mediante comandos de voz, dejando atrás interfaces mediante botones o pantallas táctiles.

El modelo de Ciudad Inteligente se encuentra en las agendas de todos los gobiernos desarrollados del mundo, estando ya establecido en las ciudades más importantes del mundo. La sobrepoblación de las urbes y la necesidad de encontrar una manera eficiente de gestionar los recursos de esta requieren un modelo a la altura.

Este trabajo pretender unificar ambas tecnologías, permitiendo al usuario el acceso a información en tiempo real y el control de plataformas de ciudades inteligentes a través de asistentes virtuales basados en voz.

Abstract

Virtual assistants are offered to us as one of the revolutionary technologies that exist nowadays and the basis for human-machine interaction in the systems of the future. This technology allows the development of more user-friendly systems, as we can find in smart homes, allowing the management of almost any electronic device through voice commands, leaving behind interfaces through buttons or touch screens.

The Intelligent City model is on the agendas of all developed governments in the world, being already established in the most important cities in the world. The overpopulation of cities and the need of finding an efficient way to manage their resources requires a model that is up to the task.

This work aims to unify both technologies, allowing the user to access information in real time and control the Smart Cities platforms through voice-based virtual assistants.

Índice general

Índice de figuras

Lista de acrónimos

ADC	Alexa Developer Console
API	Application Programming Interface
APL	Alexa Presentation Language
ASK	Alexa Skills Kit
ASR	Automated Speech Recognition
AVS	Alexa Voice Service
AWS	Amazon Web Services
CALO	Cognitive Assistant who Learns and Organizes
CGLU	Ciudades y Gobiernos Locales Unidos
DARPA	Defense Advanced Research Projects Agency
GAFAM	Google – Apple – Facebook – Amazon – Microsoft
GE	Generic Enablers
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
I+D	Investigación + Desarrollo
IoT	Internet of Things
JSON	JavaScript Object Notation
NLU	Natural Language Understanding
OCB	Orion Context Broker
PEP	Plataforma de Edificación Passivhaus
REST	Representational State Transfer
SAN	Subject Alternative Name
SRI	Stanford Research Institute
SSL	Secure Socket Layer
TIC	Tecnologías de la Información y la Comunicación

TLS Transport Layer Security

URL Uniform Resource Locator

VAVA Voice Activated Virtual Assistant

1 Introducción

Hoy en día tenemos a nuestro alcance numerosas tecnologías empleadas para mejorar nuestro día a día. En esta línea se encuentran los dispositivos de Internet of Things (IoT) que se encuentran en las Ciudades Inteligentes o Smart Cities.

No toda la ciudadanía sabe la cantidad de información de uso público que puede conocer haciendo uso de estas tecnologías. Pero hay que ser conscientes de que poder acceder a esa información de una manera amigable es esencial para acercar este nuevo ecosistema al usuario común.

1.1 Motivación

Todos hemos hecho uso de aplicaciones o dispositivos para conocer la temperatura o algún otro dato meteorológico que tiene lugar actualmente en nuestra ciudad. Sin embargo, esta información puede que sea algo escasa cuando sabemos que, gracias a sensores repartidos por toda nuestra ciudad podemos conocer la temperatura exacta en un lugar concreto, pudiendo haber variaciones entre dos puntos de la ciudad que se encuentren alejados.

El problema se presenta cuando esta información no es fácilmente accesible para el ciudadano común, bien por desconocimiento de su existencia o por dificultad en el acceso a ella. La amigabilidad y facilidad para el usuario determina el uso que el ciudadano hará de todo ello y su grado de satisfacción.

Por esa razón, se plantea una solución que introduce otra de las grandes tecnologías del presente y del futuro, los asistentes virtuales. Son varias las grandes empresas tecnológicas que han apostado por esta tecnología en la última década, y viendo el éxito que han tenido, podemos asegurar que no se equivocaron. [1]

La tecnología fue creada para servir al ser humano y hacerle la vida más fácil, o como se suele decir, que “la tecnología trabaje para el ser humano”. Juntando estos dos avances tecnológicos, además de eso, se consigue que la tecnología trabaje para la tecnología.

1.2 Objetivos

El objetivo principal de este trabajo consiste en la integración de un asistente virtual mediante sistema de voz para el uso de información obtenida mediante dispositivos IoT.

Además de este objetivo principal surgen otros objetivos, algunos de tipo más técnico, que han sido planteados y fijados para la consecución del proyecto. Esos objetivos se enumeran a continuación:

- Mejorar la amigabilidad de los sistemas IoT empleados en las Ciudades Inteligentes para facilitar la información a los usuarios.

- Investigar sobre el funcionamiento de sistemas de IoT en las Ciudades Inteligentes, concretamente en la ciudad de Santander.

- Crear y desarrollar una skill de Alexa, primero mediante los recursos que proporciona Amazon y después independizándose de dichas herramientas para depender lo mínimo posible de terceros.

- Adquirir nuevos conocimientos técnicos, especialmente de programación y de servicios web REST.

1.3 Estructura del trabajo

El trabajo se divide en seis capítulos. El primero de ellos es esta breve introducción en la que se presenta la razón de ser del proyecto y una puesta en contexto. Tras él se encuentra el capítulo *Asistentes virtuales y sistemas de voz*, en la que se explica la importancia de estos sistemas en la actualidad y en el futuro, además de la tecnología base de estos sistemas, que es el reconocimiento de voz. También se comenta la evolución que han tenido los asistentes virtuales en la última década, y se presentan los grandes sistemas que se encuentran en el mercado.

El tercer capítulo, *Plataformas IoT para Ciudades Inteligentes* consiste en la presentación de los sistemas IoT en las Ciudades Inteligentes. Se analizan los distintos tipos de dispositivos IoT que encontramos en estos sistemas. Además, se centra el estudio en la plataforma de ciudad inteligente de la ciudad de Santander, Smart Santander, en su funcionamiento y otras características relevantes. La última parte de este capítulo se dedicará a profundizar en la plataforma FIWARE como plataforma genérica habilitadora de gestión de la información en ciudades inteligentes.

En el cuarto capítulo, llamado *Alexa Skill: diseño*, nos centraremos en la explicación del desarrollo de skills con las herramientas que son proporcionadas por Amazon y en ese mismo desarrollo de una manera más independiente. Se analizan las diferencias entre ambos métodos y las ventajas e inconvenientes de uno u otro.

El quinto capítulo, *Alexa Skill: desarrollo*, se dedica al análisis en profundidad de la propuesta creada en el proyecto. Se desarrolla desde el punto de vista de integración de los sistemas analizados en los anteriores capítulos.

Por último, en el último capítulo se esbozan unas líneas de análisis general del proyecto y líneas futuras sobre el sistema creado y sobre las tecnologías empleadas.

2 Asistentes virtuales y sistemas de voz

Durante la última década estamos viviendo una época dorada en el sector de la tecnología. Hemos visto el origen de grandes avances como el machine learning, la inteligencia artificial, los sistemas de conducción autónoma y un largo etcétera. Entre estos avances se encuentra el mundo de los asistentes virtuales y los sistemas de reconocimiento de voz. Este capítulo está destinado al estudio y análisis de esta tecnología, desde el punto de vista de la relación máquina-humano.

2.1 Importancia de los sistemas de voz y asistentes virtuales

Todo el mundo, vinculado o no con el sector tecnológico, conoce lo que significa el acrónimo GAFAM, aunque tal vez no lo conozcamos de esa manera. GAFAM es una forma de referirse a las que seguramente sean las cinco empresas tecnológicas más importantes del mundo hoy en día: Google, Apple, Facebook, Amazon y Microsoft. Si nos fijamos en la relación que tienen estas empresas con el mundo de los asistentes virtuales y los sistemas de voz, veremos que cuatro de ellas tienen su propio asistente virtual. Esta apuesta deja clara la importancia de la tecnología con la que se trabaja en el presente trabajo en el mundo actual.

Si nos fijamos en la forma de relacionarnos con las máquinas, veremos que el reconocimiento de voz está desplazando a otras tecnologías como puede ser la interacción mediante botones físicos, que puede que hoy en día veamos ya algo lejana, o la funcionalidad táctil, la gran reina de la interacción humano-máquina (Figura *2.1*). Gracias a las tecnologías de reconocimiento vocal nos es posible relacionarnos e interaccionar con las máquinas únicamente usando la voz.

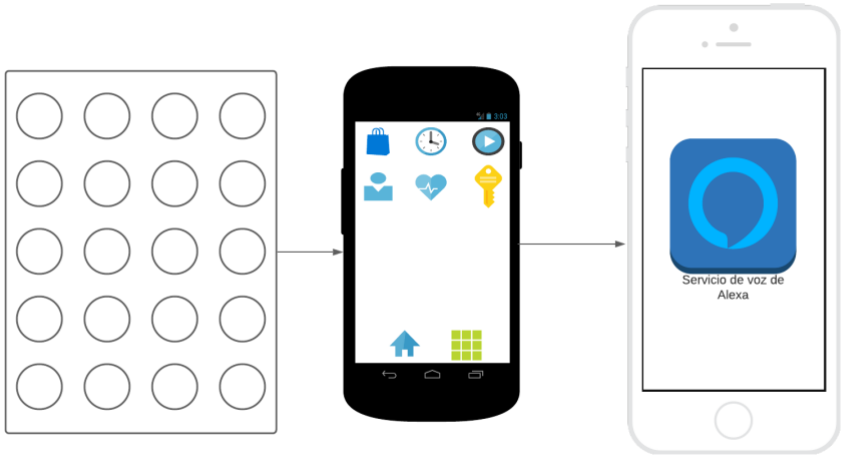


Figura 2.1: Evolución relación humano-máquina

En menos de una década, estos Asistentes Virtuales Activados por Voz (VAVA, *Voice Activated Virtual Assistant)* han crecido en número, variedad y visibilidad. El primer VAVA que se dio a conocer y, tal vez, el más mediático fue *Siri*, el asistente de Apple, cuyo lanzamiento fue en 2011. En el apartado 2.3 se hablará más detalladamente de *Siri* y del resto de asistentes de las grandes tecnológicas. Google fue la siguiente empresa en mover ficha presentando *Google Now* en 2012, el antecesor del *Google Assistant* que se dio a conocer en 2016. Microsoft, por su parte, presentó *Cortana* en 2013, que se lanzaría finalmente en 2015. Alexa, que será analizada en profundidad en los capítulos 4 y 5, fue presentada por Amazon en 2015, únicamente en Estados Unidos. Esta presentación trajo consigo el lanzamiento de los Amazon Echo, relevantes porque Alexa se convertiría en el primer VAVA vinculado a un dispositivo doméstico independiente en lugar de integrarse en uno ya existente, como por ejemplo un terminal móvil. [2]

Al igual que muchos otros avances tecnológicos, como es el caso de uno de los inventos más importantes del ser humano, Internet, los asistentes virtuales también tienen un origen militar. El padre uniformado de los asistentes virtuales se llamó proyecto CALO (*Cognitive Assistant who Learns and Organizes*) [3] vinculado a la Agencia de Proyectos de Investigación Avanzados de Defensa de EEUU (DARPA, *Defense Advanced Research Projects Agency)*. Este proyecto fue desarrollado entre 2003 y 2008, reuniendo a más de 300 investigadores con una inversión de 150 millones de dólares.

Las tareas que realizaba el sistema CALO consistían en la organización de información de correos electrónicos, documentos, calendarios y proyectos del usuario y tomar notas y organizar agendas de reuniones, es decir, realizaba las funcionalidades de un secretario digital. Además de esto, también pretendían que CALO aprendiera a interpretar las preferencias de los usuarios para modificar sus acciones en base a estas, lo que más adelante terminaría siendo lo que define a los asistentes virtuales, la personalización.

El pryecto CALO llegó a su fin en 2008, pero SRI International continuó la investigación civil siguiendo la experiencia de CALO. El SRI es un instituto científico de investigación sin ánimo de lucro que fue fundado como el Stanford Resarch Institute, pero ya en 1970 se separó de Stanford y, desde 1977, es conocido como SRI International. El SRI fundó la empresa Siri Inc. para seguir con el estudio de los asistentes virtuales, hasta que en 2010 Apple compró la empresa para incorporar el asistente virtual Siri a sus iPhone.

Esta integración con los teléfonos inteligentes fue clave para el desarrollo de los asistentes virtuales, ya que gracias a ello se dio acceso a estos sistemas para acceder a una gran cantidad de datos con los que personalizar los algoritmos y poder entrenarles.

2.2 Reconocimiento de lenguaje humano

Ya conocemos la importancia de estos sistemas de voz, pero nos queda entender cómo realmente funcionan, cómo recogen la información por parte del usuario y cómo son capaces de elaborar una respuesta.

Lo primero que deben hacer estos sistemas y la base de su funcionamiento es que deben ser capaces de reconocer la secuencia de palabras pronunciadas por el usuario [4]. A esto nos referimos con el término de reconocimiento de lenguaje humano. Este es el primer obstáculo a la hora de desarrollarlos, por la complejidad de esa fuente de información que es el habla.

Cada persona tiene una manera distinta de hablar. Por lo tanto, partimos de la base de que la voz es una característica diferenciadora en el ser humano. Esto supone un reto a la hora de identificar el contenido de una conversación o frase vocalizada, puesto que la misma secuencia de palabras puede ser pronunciada de infinitas maneras distintas.

Estos problemas o retos que supone el analizar y comprender el lenguaje natural para las máquinas van a ser analizados en el siguiente subapartado.

2.2.1 Dificultades en el procesamiento del lenguaje natural

Para comprender la dificultad del lenguaje podemos hacer referencia a una cita del artista chileno Alejandro Jodorowsky. [5]

> *"Entre lo que pienso, lo que quiero decir, lo que creo decir, lo que digo, lo que quieres oír, lo que oyes, lo que crees entender, lo que quieres entender y lo que entiendes, existen nueve posibilidades de no entenderse."*

Esta afirmación nos indica que son múltiples las dificultades que nos encontramos a la hora de procesar el lenguaje humano. Pensemos en los problemas de comunicación que pueden surgir entre seres humanos y hagámonos una idea de la dificultad que supone eso para las máquinas.

A continuación, van a ser analizadas las principales dificultades asociadas a la detección y procesamiento del lenguaje natural. [6]

2.2.1.1 Ambigüedad

El lenguaje natural es extremadamente ambiguo, como ya ha sido recalcado varias veces en los párrafos anteriores, y esta ambigüedad se da en los diferentes niveles del lenguaje.

- En el nivel léxico tenemos que una palabra puede tener varios significados. La selección del correcto se obtiene a partir del contexto de la oración.

- A nivel sintáctico nos encontramos con la resolución de anáforas y catáforas.

- En el nivel pragmático tenemos que, a menudo, una oración no significa lo que realmente se está diciendo. Aquí entran en juego elementos como la ironía o el sarcasmo, entre otros.

Para salvar estos problemas, lo que hacen los sistemas de reconocimiento de lenguaje natural es traducir ese lenguaje natural a un lenguaje semántico que pueda ser procesado por ellos.

2.2.1.2 Detección de separación entre palabras

Los seres humanos al hablar no hacemos pausas entre las palabras, sino que pronunciamos cada oración de forma continua. Es nuestro cerebro el que, al recibir la información que escuchamos, separa esas palabras dándole a esa información un sentido lógico, tanto gramatical como contextual.

Los sistemas de reconocimiento automático del habla también tienen que hacer esa labor.

2.2.1.3 Recepción imperfecta de datos

Dejes en el habla, regionalismos o acentos extranjeros son algunos de los inconvenientes que se encuentran los sistemas de reconocimiento de voz a la hora de recoger con exactitud la información hablada que se les proporciona. También hay palabras que suenan de forma similar en un mismo idioma, como es el caso de *ay* y *hay*.

Otros problemas más allá del habla pueden ser problemas tecnológicos (calidad del micrófono), que haya varias personas hablando a la vez, que haya ruido ambiente…

2.2.2 Arquitectura por niveles o capas

Anteriormente ya ha sido comentado que el lenguaje se estructura en una serie de niveles. Estos niveles son los mismos que definen la arquitectura de un sistema de reconocimiento automático del habla. [7]

Esta estructuración en niveles corresponde, de una forma similar, a la que nuestro cerebro realiza en el procesamiento del lenguaje. Partiendo de la comprensión de las unidades más básicas del lenguaje, los sonidos, se llega hasta la contextualización de las oraciones dotándolas de sentido.

- Nivel fonológico: en este nivel se realiza el reconocimiento de los fonemas, es decir, captar los sonidos y relacionarlos con las palabras que forman para transformarlos en datos procesables por el sistema. Muchos de los problemas comentados en el apartado anterior se dan principalmente en este nivel, y por esa razón es que este nivel es dependiente de los superiores.

- Nivel morfológico: trata de analizar las palabras anteriores para dotarlas de un sentido de manera aislada, es decir, dotar de sentido a nivel de palabra. Para realizar este análisis es necesario que el sistema sea capaz de analizar los monemas, la unidad del lenguaje situada por debajo de la palabra.

- Nivel sintáctico: este nivel se centra en el análisis de las relaciones entre los grupos de palabras, y es donde se constituye la gramática de la frase.

- Nivel semántico: aquí se constituye el significado aislado de cada oración, sin tener en cuenta el contexto de esta. Se relacionan los significados de las palabras para proporcionar ese sentido a la oración.

- Nivel pragmático: en este último nivel se proporciona ese contexto a la oración que le faltaba al nivel anterior. Para dotar de ese contexto a la oración es necesario fijarse en las oraciones que la acompañan.

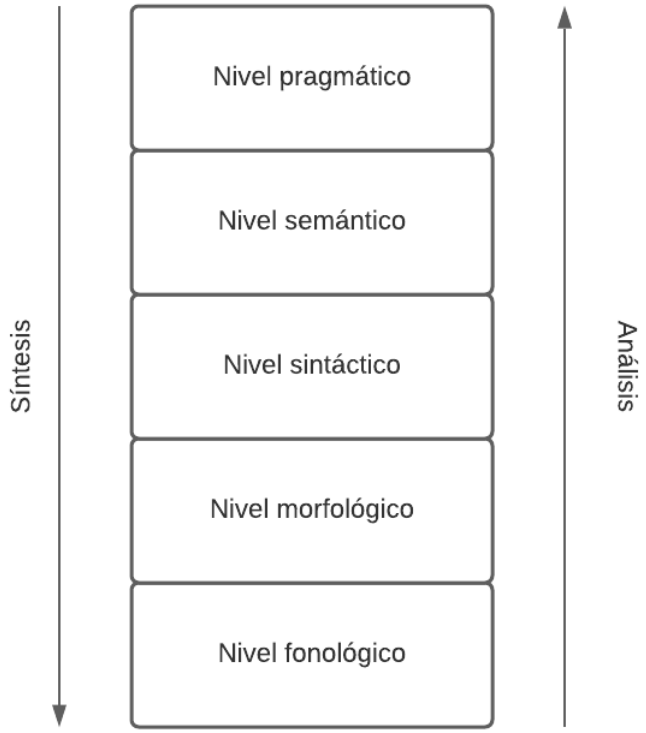


Figura 2.2: Arquitectura por niveles

Este orden por niveles, representado de forma gráfica en la Figura 2.2, por parte del sistema de reconocimiento automático del habla se realiza a la inversa a la hora de formar la respuesta.

2.2.3 Tipos de aprendizajes de un sistema de reconocimiento automático del habla

Además de este funcionamiento anteriormente comentado para el análisis que siguen los sistemas de reconocimiento de voz, hay un aspecto muy importante que aparece a la hora del diseño de estos, y es la elección del tipo de aprendizaje que se va a utilizar para construir las diversas fuentes de conocimiento. [8]

- Aprendizaje deductivo: este tipo de aprendizaje se basa en la transferencia de conocimientos por parte del ser humano hacia el sistema

informático. Un ejemplo de este tipo de aprendizaje son los Sistemas Basados en Conocimiento, es decir, aquellos sistemas en los que la forma de resolver el problema radica en poseer un conocimiento. Más concretamente, dentro de estos Sistemas Basados en Conocimiento se encuentran los Sistemas Expertos, en los que esa transferencia de conocimiento entre humano y máquina se realiza a través de la experiencia y de que el sistema emule lo que hace el humano.

- Aprendizaje inductivo: mediante esta técnica de aprendizaje, el sistema consigue por sí mismo los conocimientos necesarios para, a partir de ejemplos reales, realizar la tarea. Aquí se encuentran las redes neuronales artificiales.

En la práctica, ningún sistema está basado únicamente en solo uno de los tipos de aprendizaje, sino que se trata de un sistema híbrido de ambos en el que los aspectos generales se suministran deductivamente y la caracterización de la variabilidad inductivamente.

Pongamos un ejemplo. De forma deductiva la máquina puede aprender lo que una afirmación y que se produce cuando oye la palabra “sí”, pero inductivamente aprende que no solo puede haber una afirmación cuando escucha esa palabra, sino que hay más posibilidades. Esta mezcla de aprendizaje supone una mejora en el reconocimiento de lenguaje natural, proporcionando mucha mayor exactitud en los sistemas.

2.2.4 Clasificación de los sistemas de reconocimiento de voz

Hay varios criterios para clasificar los sistemas de reconocimiento de voz y mediante los cuales podrían ser valorados para indicar cuál tiene unas mejores prestaciones.

- Entrenabilidad: determina la necesidad del sistema de ser entrenado previamente para poder funcionar correctamente.

- Dependencia del hablante: este rasgo nos indica la independencia que tiene el sistema de reconocimiento de voz respecto del hablante. Lo más interesante y lo que nos proporcionaría una mejor experiencia de uso sería que esta independencia sea máxima, para que así el sistema fuese capaz de funcionar correctamente independientemente de quién fuese el usuario.

- Continuidad: determina la capacidad del sistema para escuchar palabras sin la necesidad de que el usuario haga pausas entre ellas. Al igual que el rasgo anterior nos interesa que esta continuidad sea máxima, porque si no la experiencia de usuario no sería nada buena.

- Robustez: esta característica nos dice si el sistema está preparado para funcionar con ruido y condiciones de audio no deseables. Cuanto mayor

sea esta robustez, menos problemas por problemas de ruido podrán suceder, lo cual provoca que el sistema no reconozca al usuario y pida numerosas veces que repita el mensaje.

- Tamaño del dominio: determina la capacidad del sistema para reconocer una cierta cantidad de palabras. Cuanto mayor sea el dominio, mayor será la capacidad del sistema para reconocer palabras diferentes. Podría decirse que es la cantidad de vocabulario de la que dispone el sistema.

2.3 Asistentes virtuales más destacados

Anteriormente se ha mencionado el término GAFAM, para referirse a las empresas Google, Apple, Facebook, Amazon y Microsoft, y se ha comentado que cuatro de ellas han desarrollado su propio asistente virtual.

En este apartado del capítulo se va a profundizar un poco más en cada uno de esos cuatro asistentes virtuales.

2.3.1 Siri

Siri fue el primer asistente virtual por reconocimiento de voz que se lanzó al mercado, y además de ser el primero, es considerado el más importante de esta lista. Siri es el asistente virtual que podemos encontrar en todos los dispositivos del ecosistema Apple, desde iPhone hasta iMac, pasando por watchOS, iPad y Macbook.

Las funcionalidades que tiene Siri son infinitas. Entre las más importantes destacan realizar llamadas, mandar mensajes, recordatorios, recomendaciones de tráfico, programar el despertador, poner música y, lo más destacable, preguntar sobre todo tipo de cosas. Siri tiene respuesta prácticamente para todo, y para todo aquello que no lo tenga realizará una búsqueda en el navegador que será mostrada por pantalla.

A la hora de desarrollar herramientas con Siri tenemos SiriKit [9], el *framework* que permite integrar el asistente en apps de terceros. Por lo tanto, esta sería la manera de integrar Siri en un producto para un desarrollador.

La forma de funcionar de este asistente virtual es muy parecida a la de los demás asistentes virtuales, la cual será analizada en profundidad en el capítulo 4 de este trabajo. El asistente reconoce el mensaje hablado que recibe para transformarlo en texto que pueda procesar para analizar la intención, es decir, saber cuál es la instrucción concreta. Posteriormente, se realiza la acción deseada con la app correspondiente, y cuando Siri obtiene la respuesta de la app se encarga de devolverle un *feedback* al usuario.

2.3.2 Google Assistant

Google Assistant [10] es el asistente virtual de Google, como su propio nombre indica, ya que es el único de esta lista que no posee un nombre propio.

Como ya hemos comentado en el caso de Siri cómo es el funcionamiento de un asistente virtual, nos centraremos en las diferencias entre cada uno de ellos. En el caso de Google Assistant encontramos la principal diferencia de la capacidad para conectar dispositivos de Smart Home. Google dispone de sus propios dispositivos, como son la serie de Google Home o la de Google Nest, que son compatibles con muchos dispositivos de hogar inteligente como bombillas, termostatos, robots de limpieza, robots de cocina…

Recientemente ha sido anunciada por parte de Google una nueva funcionalidad en su asistente virtual, el cual contará con un modo para invitados. Esto permitirá que cuando el usuario tenga un invitado en su hogar dicho invitado pueda interactuar con Google Assistant sin interferir en las preferencias del usuario propietario y sin interferir en las preferencias guardadas por el asistente. [11]

2.3.3 Cortana

Cortana es el asistente virtual de Microsoft. La forma de funcionar es la más parecida a Alexa, que es el asistente virtual empleado para este trabajo y que se explicará en profundidad en los capítulos 4 y 5. Opera a través de lo que se conoce como *skills*. Las skills son habilidades o tareas que el asistente puede realizar. Si queremos que Cortana realice algún tipo de funcionalidad deberemos añadir la skill a nuestro sistema, o bien desarrollar nuestra propia skill.

Cortana Skill Kit [12] es la forma de poder desarrollar nuestras propias skills para poder conectar a los usuarios con nuestros servicios personalizados.

2.3.4 Alexa

Por último, tenemos a Alexa, el asistente virtual de Amazon integrado en todos los dispositivos de la serie Echo. Este asistente virtual va a ser analizado en profundidad más adelante con numerosos ejemplos prácticos. Por el momento hay que indicar que el funcionamiento es similar al de Cortana. También se basa en *skills* y el *framework* para ello es llamado Alexa Skills Kit. [13]

La skills se lanzan mediante un *Skill Invocation Name*, al cual irá vinculado lo que pretendamos que haga la skill. Lo que podemos decir estará definido en el *Interaction Model*, el cual define qué es lo que va a recibir la skill, la intención de la orden y la información que debe procesar.

3 Plataformas IoT para Ciudades Inteligentes

Durante los últimos años está a la orden del día el concepto de Ciudad Inteligente. Por numerosas razones que serán analizadas a continuación, se busca adaptar la ciudad a las nuevas necesidades, buscando la sosteniblidad.

Las ciudades inteligentes han centrado los esfuerzos de organismos públicos y empresas privadas. Este trabajo es mi aportación a este mundo para acercar aún más este modelo de ciudad al ciudadano, uniéndolo a la tecnología de asistentes virtuales que se ha introducido en el capítulo 2. Los conceptos que se desarrollan en este capítulo serán claves para entender el objetivo del trabajo.

A continuación, se verá la razón de ser de las ciudades inteligentes, su forma de funcionar y cuál es el objetivo de estas. Es decir, el por qué, cómo y para qué.

3.1 Ciudades Inteligentes

Una ciudad inteligente o *Smart City* es aquella ciudad que pretende mejorar la vida de sus ciudadanos apostando por la sostenibilidad, es decir, por la gestión eficiente en todos los ámbitos de aplicación de la ciudad.

Los ámbitos de aplicación de las ciudades inteligentes son muy amplios, siendo los siguientes los más destacables, junto con algunos ejemplos de elementos promovidos:

- Urbanismo: sistemas de alumbrado público con nuevas tecnologías y adaptaciones al consumo, riego de jardines públicos, construcción de edificios sostenibles.

- Transporte y movilidad: gestión eficiente del tráfico, optimización de las rutas del transporte público para la reducción de su huella de carbono, gestión del aparcamiento, potenciar las tecnologías de coches eléctricos con puntos de carga.

- Sanidad: mejoras en la gestión de datos e historiales de pacientes, teleasistencia, mejora de los servicios de alerta a los servicios de emergencia.

- Seguridad pública: mejora de la respuesta de los servicios de seguridad ciudadana.

- Energía y medio ambiente: consumo eficiente del agua, reducción de las emisiones de gases contaminantes, fomento de tecnologías renovables, sistemas que permitan el ahorro de energía.

- Administración pública: administración electrónica, conexión wifi en toda la ciudad.

- Turismo y ocio: facilidades en la búsqueda de restaurantes, centros culturales y centros de ocio, guías turísticas electrónicas permitiendo la personalización a los intereses del usuario.

Para conseguir esto es necesario reinventar los modelos de habitabilidad de las ciudades, porque estas cada vez tienen mayor población y, del mismo modo, aumenta la demanda de necesidades y servicios. Este crecimiento de población se está dando de manera exponencial, y donde más crece es en los núcleos urbanos.

El crecimiento es tal que se estima que en el año 2050 el 70% de la población mundial vivirá en ciudades [14]. Esta atracción de población hacia las ciudades está motivada por la concentración en estas de los centros de negocio y de ocio y cultura.

España tiene una situación particular, demográficamente hablando. Según el Instituto Nacional de Estadística en 2019, existen 63 ciudades en España con más de 100.000 habitantes, y hasta 148 con más de 50.000 [14]. Este número es mayor que el de países con mayor población, como es Francia, hasta el punto de que Madrid es la tercera ciudad más poblada de Europa Occidental. En contraposición, de los 8.166 municipios españoles, cerca del 60% no llega al millar de habitantes y el 13% no llega ni siquiera a 100. Con estos datos nos podemos hacer una idea de la concentración de población en las ciudades españolas

3.1.1 Razones para impulsar las Smart Cities

Una de las razones principales por las que impulsar las ciudades inteligentes es el problema demográfico, pero a este hay que sumarle otras dos como son el aumento del consumo energético y el aumento de la contaminación. Esta terna conforma un tridente de razones correlativas.

Según datos que aporta la Plataforma de Edificación Passivhaus (PEP) en 2019, los vehículos producen el 13% de la contaminación en las ciudades, mientras que las viviendas y edificios el 56% [15]. Este dato nos permite comprender la importancia de la eficiencia energética en hogares para poder asumir el crecimiento en las ciudades que se viene dando en los últimos años, además del crecimiento que se prevé en un futuro cercano.

Otro dato que aporta la PEP es que la vida útil de un vehículo es de aproximadamente 10 años, mientras que la de las viviendas es de hasta 40 años. Por lo tanto, tenemos en las viviendas un gran consumidor de energía durante un largo periodo de tiempo, y de ahí la importancia de construir casas que sean eficientes desde el punto de vista energético. Las ciudades inteligentes permiten que las casas ya construidas puedan adaptarse a estos nuevos tiempos para evitar ese agujero de gasto energético y contaminación.

3.1.2 Principales barreras y elementos facilitadores

Para poder analizar el modelo de Smart City es necesario comprender que herramientas y aspectos favorecen el desarrollo de este tipo de proyectos, además de que dificultades se van a encontrar y van a requerir del estudio de una solución frente a ellas.

La Comunidad de Práctica de Ciudades Digitales de CGLU (Ciudades y Gobiernos Locales Unidos) viene realizando cada pocos años un estudio, llamado *Smart Cities Study*, en el que se analiza la situación actual de las ciudades inteligentes. Recientemente se ha publicado la tercera edición de 2019, pero en la de 2017 es en la que se encuentran estos aspectos que pueden favorecer o perjudicar a las Smart Cities. Este estudio [16] se realiza para ofrecer información sobre los diferentes proyectos de ciudad inteligente que se están llevando a cabo en el mundo.

Las principales barreras que pueden encontrarse a la hora del desarrollo de proyectos de Smart City se pueden dividir en tres grandes bloques.

1. Innovación, emprendimiento y generación de actividad económica.
    - El bajo espíritu emprendedor.
    - La dificultad de acceso a financiación por parte de los proyectos emprendedores (especialmente de aquellos mecanismos de financiación más adecuados, como capital riesgo, capital semilla…).
    - La insuficiente conexión entre las universidades y el sector privado.
2. Conocimiento y talento.
    - La baja conexión entre la Universidad y las Administraciones públicas.
    - La dificultad de atracción de talento externo.
3. Sociedad y economía digital.
    - La complejidad de los procesos burocráticos existentes a nivel administrativo en cada ciudad.
    - La falta de alineamiento entre los diferentes agentes públicos
4. Otros.
    - La falta de contacto con otras experiencias internacionales.
    - Contar con un marco legislativo no actualizado.

Por otro lado, también han sido identificados elementos facilitadores para el desarrollo de este tipo de proyectos, a nivel privado y público, recordando la importancia de ambos sectores para que esta idea de ciudad inteligente funcione.

Estos aspectos beneficiosos se agrupan en los tres ámbitos comentados anteriormente.

1. Innovación, emprendimiento y generación de actividad económica.
    - La existencia de infraestructuras para el desarrollo de proyectos emprendedores.
    - La existencia de mecanismos flexibles de financiación.
    - El apoyo de la I+D y, en concreto, a la generación de patentes y prototipos.
    - El acompañamiento y apoyo técnico a las personas emprendedoras.
2. Conocimiento y talento.
    - El desarrollo de programas de partenariado entre la universidad, la administración y el sector privado.
3. Sociedad y economía digital.
    - La existencia de una planificación a distintos niveles enfocada a la promoción del emprendimiento.
    - La transparencia y la disponibilidad de información libre (open data).
    - El alineamiento de las distintas administraciones públicas en torno a agendas comunes de apoyo al emprendimiento y la innovación.

3.2 Dispositivos IoT

Hasta ahora se han descrito las bases y necesidades en las que se sustentan las ciudades inteligentes, pero aún no se ha comentado como se obtiene la cantidad ingente de datos e información que se necesita para hacerlas realidad.

El término IoT corresponde a las siglas en inglés de *Internet of Things*, y que mejor definición del término que la de quien propuso este concepto en 1999, Kevin Ashton. [17]

> *"Los ordenadores actuales y, por tanto, Internet son prácticamente dependientes de los seres humanos para recabar información. El problema es que las personas tienen tiempo, atención y precisión limitados, lo que significa que no son muy buenos a la hora de conseguir información sobre cosas en el mundo real. Y eso es un gran obstáculo.*
>
> *La tecnología de la información actual es tan dependiente de los datos escritos por personas que nuestros ordenadores saben más sobre ideas que sobre cosas. Si tuviéramos ordenadores que supieran sobre las*

"cosas", mediante el uso de datos que ellos mismos pudieran recoger sin nuestra ayuda, nosotros podríamos monitorizar, contar y localizar todo a nuestro alrededor. Sabríamos cuando reemplazar, reparar o recuperar lo que fuera, así como conocer su funcionamiento.

El Internet de las Cosas tiene el potencial para cambiar el mundo tal y como hizo la revolución digital hace unas décadas. Tal vez incluso hasta más."

Algo más de dos décadas después no podemos saber si Kevin Ashton tiene razón en todo lo que dijo, pero podemos asegurar que va por el buen camino.

Los dispositivos IoT nos permiten poder obtener información de nuestro entorno las 24 horas del día los 365 días del año, algo esencial dentro del concepto de Smart City.

El concepto clave de estos dispositivos es que inicialmente no son objetos que relacionamos con Internet, pero que dotados de la tecnología necesaria pueden hacerlo para proporcionarnos la información deseada. Con esto quiero decir que lo que pretende la IoT es extender Internet más allá de los dispositivos tradicionales para poder estar presente en otros dispositivos que puedan ser controlados de forma remota.

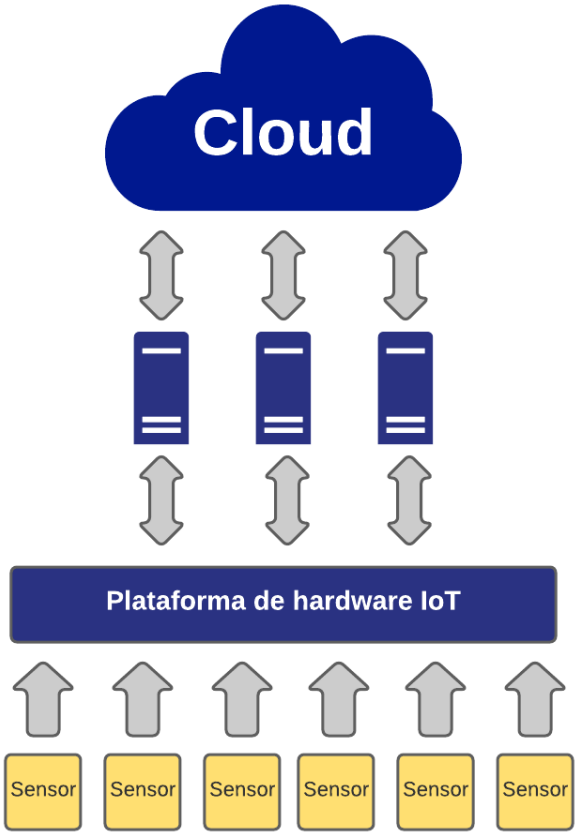


Figura 3.1: Arquitectura de sistema IoT

El conjunto de dispositivos IoT están mayormente formado por sensores, cuyo hardware se encarga de la monitorización, medición y recolección de datos. La función principal de un sensor es convertir un fenómeno físico en una señal que pueda ser medida y que se puede transmitir inalámbricamente, como suele ser lo más común, o mediante señales eléctricas u ópticas. Pero los sensores empleados en sistema IoT no son simples sensores, ya que se les suele asociar la etiqueta de inteligentes. Esta etiqueta corresponde a que esta categoría de sensores son la combinación de un sensor, convertidor de señal analógica a digital, un microprocesador para procesar información o recibir órdenes, y una interfaz de comunicación. Las mejoras en tecnología han permitido abaratar los costes de estos dispositivos para aumentar su despliegue.

A continuación, van a ser presentados los distintos tipos de sensores inteligentes que se encuentran en las ciudades inteligentes, siendo los dos primeros de ellos los más importantes y empleados en el desarrollo de este trabajo.

Además de todos los sensores que se van a enumerar, hay muchos otros que también están presentes en las ciudades inteligentes, como son los medidores de CO_2 y otros medidores de calidad del aire o los de luminosidad.

En la Figura 3.1 se puede ver como funcionan el conjunto de sensores que forman un sistema de IoT, guardando la información que recogen en unos servidores accesibles desde la nube.

3.2.1 Sensores de temperatura

Estos sensores se pueden emplear en numerosos entornos de aplicación de tecnología IoT, desde entornos abiertos como las ciudades inteligentes hasta edificios como puede ser el caso de oficinas, centros escolares o entornos industriales. Además, no solo tiene por qué medir la temperatura ambiente. Por ejemplo, en un entorno industrial puede medir la temperatura de las máquinas para comprobar su correcto funcionamiento.

Una posibilidad que nos ofrecen este tipo de sensores es la de combinarlos con otro tipo de hardware, los actuadores, para así, por ejemplo, influir sobre un termostato en el caso de que la temperatura ambiente no se encuentre en el rango deseado.

3.2.2 Sensores de humedad

Este tipo de sensor es muy similar al del caso anterior. Se pueden emplear para medir la humedad relativa del aire en una determinada área, aunque el tipo de sensor de humedad que viene siendo más común últimamente es el de suelo. Este sensor es empleado por productores agrícolas para medir y controlar las tasas de humedad de una plantación, realizando así un consumo más eficiente de los recursos del agua, pudiendo instaurar un accionamiento de riego automático cuando el nivel de humedad del suelo se sitúe por debajo de un umbral.

3.2.3 Sensores de proximidad

Muy comúnmente usados para el monitoreo y disponibilidad de lugares de estacionamiento en grandes espacios como aeropuertos o centros comerciales, este tipo de sensor se está incluyendo también en el concepto de Smart City para controlar esa disponibilidad de estacionamiento en toda la ciudad.

3.2.4 Acelerómetro y giroscopio

Son usados para detectar vibraciones, inclinación y aceleración lineal. Dentro de las Smart Cities es usado, por ejemplo, en los medios de transporte público. Conociendo a la velocidad que se mueve un autobús urbano, por ejemplo, se pueden hacer cálculos en base a su posición, al tráfico y la ruta a seguir para conocer los tiempos entre las diferentes paradas

3.2.5 Sensores de nivel

Estos sensores permiten monitorear el nivel de líquidos y otros fluidos dentro de un recipiente. Son muy usados en la gestión de residuos, permitiendo una recogida eficiente de estos en cada punto habilitado para ello. De este modo, el servicio encargado del vaciado de los contenedores solo pasará por los que estén cercanos a llenarse.

3.3 Smart Santander

Smart Santander [18] es el nombre del proyecto de ciudad inteligente que se comenzó a desarrollar a principios de la década pasada en la ciudad de Santander. El proyecto ha estado liderado por Telefónica y la Universidad de Cantabria, contando con el apoyo del Gobierno de Cantabria y el Ayuntamiento de Santander. El objetivo de este proyecto ha sido el de convertir a Santander en una ciudad inteligente y mejorar la calidad de vida de los ciudadanos por medio de la innovación tecnológica de IoT.

Este proyecto supuso el despliegue de 20.000 dispositivos de IoT, entre sensores, actuadores, captadores y terminales móviles, entre otros. Todo esto financiado bajo un presupuesto de 8,67 millones de euros.

La información captada por la plataforma de Smart Santander es de libre acceso, y en el capítulo 5.3.1 se explica cómo es posible poder consultar dicha información. En la documentación de la API [18], se pueden consultar los distintos tipos de fenómenos físicos que se pueden captar, entre los que destacan numerosas características para conocer la calidad del aire, datos de tráfico, humedad y temperatura ambiental y otros tipos de datos meteorológicos, además también de gran cantidad de datos sobre el estado de servicios públicos, como el de los contenedores de residuos o de jardines públicos.

Además de los distintos fenómenos medibles también se encuentra entre la documentación las unidades de medida de todos estos tipos de fenómenos, para facilitar a los desarrolladores su labor en la creación de nuevas aplicaciones y servicios empleando los datos proporcionados por Smart Santander.

3.4 FIWARE

FIWARE [19] es una plataforma creada para el impulso del desarrollo de plataformas de Internet del Futuro. Es una plataforma abierta, pública y libre cuya misión es "construir un ecosistema sostenible abierto en torno a estándares de plataforma de software públicos, libres de regalías y basados en la implementación que facilitarán el desarrollo de nuevas aplicaciones inteligentes en múltiples sectores".

Esta plataforma fue impulsada por la Unión Europea, financiada por VII Programa Marco, dentro de su proyecto de colaboración público – privada para el Internet del Futuro.

FIWARE es una iniciativa que define un conjunto universal de estándares para la gestión de datos contextuales que facilitan el desarrollo de soluciones inteligentes para diferentes dominios. Proporciona capacidades en la nube basadas en código abierto junto con un conjunto de herramientas y librerías de valor añadido denominadas Generic Enablers (GE).

La Figura 3.2 muestra cómo los diferentes GE de FIWARE se combinan para conformar una arquitectura que resuelve buena parte de las necesidades de una Smart City.

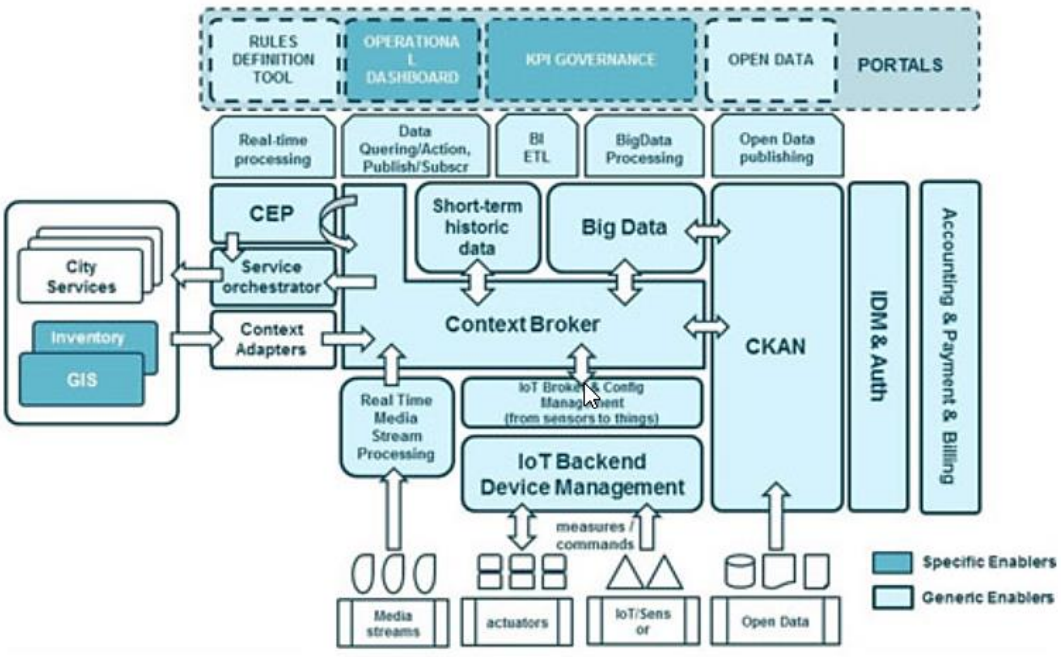


Figura 3.2: Arquitectura para Smart Cities FIWARE

El Orion Context Broker (OCB) es su componente principal, el cual aporta una función fundamental en cualquier solución inteligente, administrar la información de contexto, consultarla y actualizarla.

Esta plataforma reúne, gestiona y proporciona acceso a información contextual provenientes de diferentes fuentes que describen lo que está sucediendo en una ciudad. Un software de código abierto listo para el mercado que combina componentes que permiten la conexión a IoT con la gestión de la información contextual y los servicios de Big Data en la nube.

La plataforma FIWARE constituye la base de la infraestructura de las ciudades inteligentes y proporciona funcionalidades genéricas para soportar varios aspectos tecnológicos de las infraestructuras de Ciudad Inteligente tales como:

- Gestión de dispositivos IoT, puerta de enlace de IoT que puede conectarse con diferentes dispositivos que admiten varios protocolos de IoT y sistemas heredados.

- Gestión de datos y contexto, agente de contexto que puede administrar la información de contexto que está conectada con diferentes aplicaciones o dispositivos y proporciona información con API estándar.

- Almacenamiento de Big Data, funcionalidades para soportar el almacenamiento y la administración de gran cantidad de datos provenientes de sensores y dispositivos IoT.

- Gestión de datos abiertos, soporte para la publicación y aprovisionamiento de conjuntos de datos abiertos.

- Panel de control avanzado, posibilidad de crear interfaces de usuario (móviles) que admitan la visualización de datos en tiempo real, gráficos avanzados, cabinas, etc.

- Seguridad, administrador de identidad, control de acceso basado en roles, privacidad y anonimato.

- Nube: La plataforma debería poder ejecutar algunos de sus componentes en un entorno de nube en modo "como servicio".

Se basa en principios abiertos y bien definidos:

- Interoperabilidad, cada componente de FIWARE se puede integrar fácilmente con los demás y con los externos porque proporciona API estándar y abiertas.

- Modularidad, cada componente de FIWARE es independiente, por lo que no es obligatorio utilizar todos los componentes proporcionados en la arquitectura, pero algunos pueden sustituirse (por ejemplo, por otros propietarios relacionados con tecnologías específicas que ya están en las ciudades).

- Generalidad, una arquitectura que se puede personalizar para los diferentes dominios / casos de uso del proyecto (gestión de energía, movilidad, etc.) porque se basa en componentes genéricos.

- Reusabilidad, la plataforma se puede reutilizar fácilmente en diferentes ciudades con un esfuerzo limitado porque se basa en componentes genéricos y abiertos.

Las empresas y los desarrolladores pueden probar sus aplicaciones FIWARE en FIWARE Lab, explotando los datos abiertos publicados por ciudades y otras organizaciones. FIWARE Lab es una infraestructura de nube basada en OpenStack, un software que controla grandes grupos de recursos informáticos, de almacenamiento y de red en todo un centro de datos, administrado a través de un panel de control o mediante una API implementada en una red de nodos federados distribuidos geográficamente.

El uso de FIWARE en todo el mundo promueve un enfoque abierto, estandarizado y de mejores prácticas en el diseño e implementación de plataformas de ciudades inteligentes. Una iniciativa pionera que está presente en ciudades europeas y americanas como:

Viena (Austria), Niza y San Quintín (Francia), Génova (Italia), Utrecht (Holanda), Oporto (Portugal), Santander (España), Valencia (España), Gotemburgo (Suecia), La Plata (Argentina) y Montevideo (Uruguay).

4 Alexa Skill: diseño

Como ya fue comentado en el capítulo 2 de este trabajo, algunos asistentes virtuales como Alexa o Cortana funcionan con el concepto de skills. A lo largo de este capítulo se hablará, concretamente, de las skills de Alexa, el asistente virtual de Amazon.

Una skill es una habilidad o funcionalidad desarrollada por terceros que podemos instalar en nuestro dispositivo en el que se encuentra el asistente Alexa, que le permite a este llevar a cabo una tarea o conjunto de tareas.

Por defecto, los dispositivos Echo, que es el nombre de la serie de dispositivos de Amazon que disponen de Alexa, vienen con una serie de skills de serie, denominadas *built-in skills*. El poder crear nuestras propias skills o instalar las skills creadas por otros desarrolladores permite que nuestro asistente virtual tenga muchas más funcionalidades que las que vienen de serie.

Las skills se apoyan en un gran servicio que vive en la nube, y que está aprendiendo constantemente. En este se realizan dos procesos de *machine learning* muy importantes cuando los usuarios interactúan con una skill. La etapa inicial, llamada *Automated Speech Recognition* (ASR), es la que convierte lo que el usuario dice a texto. La otra etapa, llamada *Natural Language Understanding* (NLU) es la que le dota de sentido a ese texto obtenido de la etapa primera.

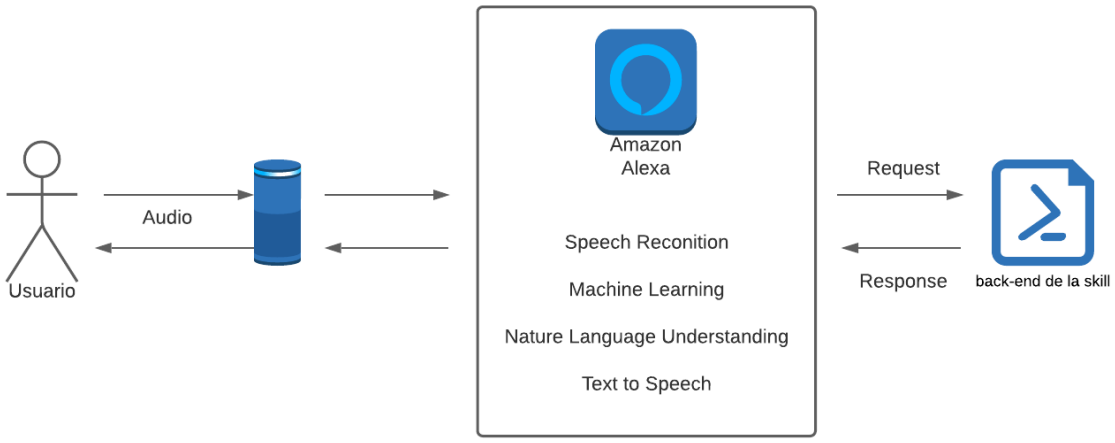


Figura 4.1: Proceso de solicitudes de Alexa

En la Figura 4.1 se puede observar como se registra y se procesa la petición que realiza un usuario a una skill. Cuando se pronuncia la *wake word* para comenzar la interacción con la skill, el dispositivo comienza a escuchar. Este dispositivo está representado en la figura como un cilindro de color azul. Cuando el dispositivo recoge la petición se realizan las etapas comentadas anteriormente para traducir dicha solicitud en una intención del usuario que poder pasar al back-end de la skill. Lo que recibe el back-end no es el audio del usuario, sino una traducción de este en forma de comando o directiva para realizar las operaciones pertinentes. Una vez armada la respuesta, en modo de texto, el servicio de Alexa realiza una etapa de

Text to Speech, en la que transformará ese texto en audio, el cual se hace llegar al usuario.

Las compañías que quiere incorporar Alexa a sus propios dispositivos o los usuarios que quieran obtener una skill de terceros hacen uso de *Alexa Voice Service* (AVS). Este servicio permite que la skill que cree como desarrollador puede llegar a cualquier dispositivo que lo permite (Figura 4.2).

Para poder desarrollar nuevas skills, Amazon dispone de una serie de herramientas denominada *Alexa Skills Kit* (ASK) [13]. Haciendo uso de estas herramientas y del framework que se proporciona, llamado *Alexa Developer Console* (ADC), se pueden crear y gestionar nuestras skills.

Figura 4.2: Servicios de Amazon Alexa

4.1 Alexa Developer Console

Para poder desarrollar skills es necesario es crear una cuenta de Amazon como desarrollador. A continuación, ya se puede hacer uso de la herramienta.

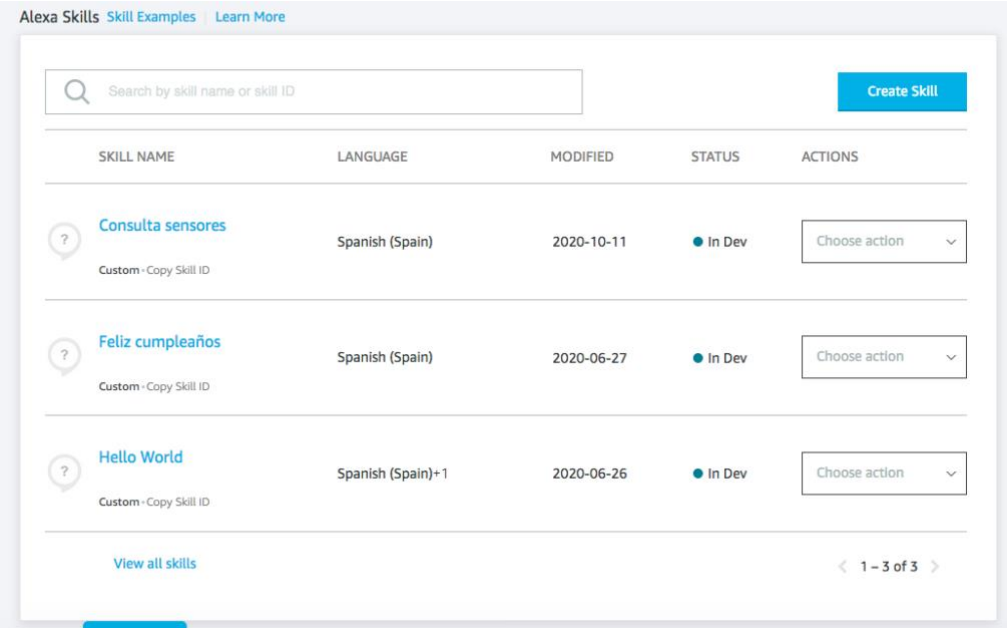


Figura 4.3: Pantalla de inicio de ADC

La skill llamada “Consulta errores”, que se puede ver en la Figura 4.3, es la que ha sido creada para este trabajo, y la que se va a analizar y desarrollar durante este capítulo y el siguiente.

Para crear una skill solo es necesario elegir la opción *Create Skill* y elegir un nombre, un *invocation name*, el modo de alojar la skill y una plantilla para comenzar a desarrollar.

Una vez hecho esto, y suponiendo que se ha elegido la opción de alojamiento proporcionada por Amazon, a la hora de desarrollar una skill mediante *Alexa Developer Console* disponemos de tres escenarios para trabajar. El primero de ellos es el *Build*, luego se encuentra *Code*, y por último aparece *Test*.

4.1.1 Build

Esta parte de ADC (Figura 4.4) es donde nos encargamos de desarrollar el front-end de nuestra skill, es decir, la parte de la skill cercana al usuario, donde se define el modelo de voz o *interaction model*, y donde se configuran los aspectos principales de la skill. Uno de estos aspectos es el *Endpoint*.

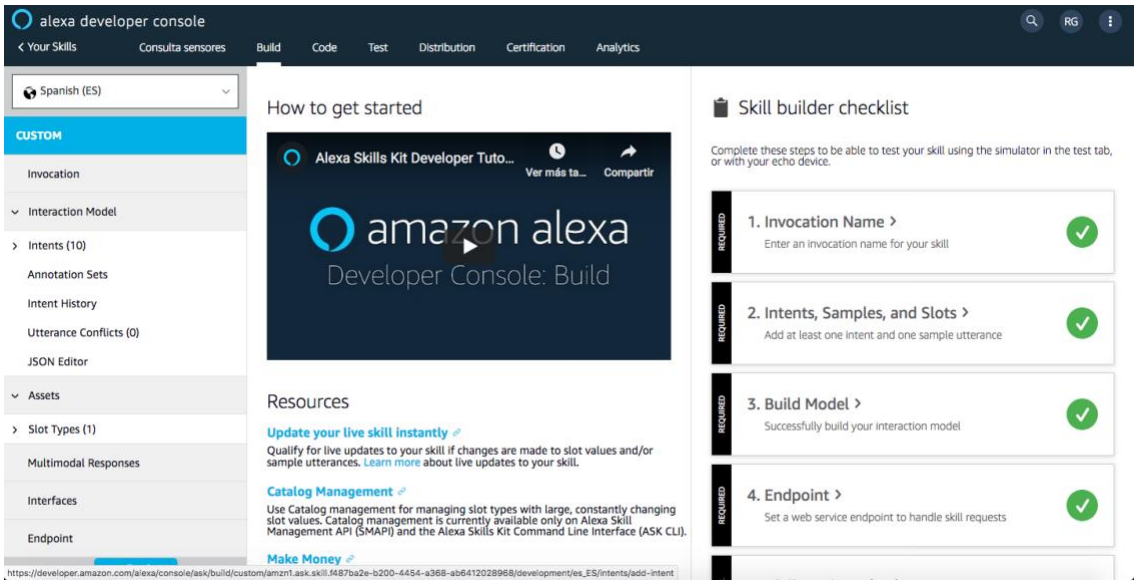


Figura 4.4: Ventana Build en ADC

4.1.1.1 Invocation

Al iniciar una skill el usuario debe pronunciar unas palabras llamadas *Invocation Name* para que el asistente Alexa sea capaz de reconocer que tipo de skill debe poner en funcionamiento para comenzar la interacción y recoger la información que venga a continuación.

Este *Invocation Name* debe constar de dos o más palabras escritas en letras minúsculas y espaciadas unas de otras. En el caso de la skill desarrollada en este trabajo cuyo nombre es “Consulta Sensores”, el *Invocation Name* también es “consulta sensores”, siendo el modo de comenzar a interactuar con la skill diciendo

“Alexa, abre consulta sensores”. Es posible cambiar la frase de lanzamiento “abre” por otras como “lanza”, “comienza”, “activa” o “carga”.

Cabe destacar que el *Invocation Name* no tiene por qué corresponder con el nombre de la skill. Además, hay ciertas palabras que están prohibidas en la elección de esta fórmula para abrir una skill. Algunas de estas palabras son las propias que sirven para lanzar skills como “abre”, “lanza” o “activa”, además de otras como las conocidas como *wake words*, que son “Amazon”, “Alexa” y “Echo”. Estas palabras están prohibidas porque mediante la pronunciación de las mismas se alerta al servicio de que debe comenzar a escuchar. También están prohibidas otras como “ordenador”, “skill” o “app”.

4.1.1.2 Modelo de voz

El modelo de voz o *interaction model* es la forma en la que el usuario interactúa con el asistente, y mediante este modelo se definen los diálogos que el asistente puede recibir por parte del usuario.

Para dar comienzo a una skill siempre hay que comenzar la frase con una *wake word*, mediante la cual se pone en aviso al sistema para que escuche al usuario. Por defecto esta palabra es “Alexa”. Seguida de la *wake word* irá la palabra de lanzamiento, que pueden ser términos como “abre”, “inicia”, “pide a”, “empieza”, “comienza” o “lanza”, entre muchos otros, y a continuación ya puede ir el *invocation name* (Figura 4.5).

Figura 4.5: Formulación de inicio de la skill

Para poder pedir a la skill que ejecute alguna funcionalidad de las que es capaz de realizar es necesario seguir los elementos anteriormente comentados de un *utterance*, lo que puede ser traducido al castellano como declaración o enunciado. Mediante la pronunciación de un *utterance* el usuario le da a conocer al asistente que es lo que quiere hacer. Una vez el ASR ha realizado la traducción del audio del usuario a texto, lo hace Alexa es buscar en el modelo de voz con que *intent* se corresponde ese *utterance*.

Un *intent* corresponde con una intención por parte del usuario, y una vez identificado el *intent* al que corresponde el *utterance* que ha pronunciado el usuario, el sistema de Alexa ya es capaz de realizar las acciones correspondientes para satisfacer la necesidad del usuario. Cada *intent* puede tener asociados tantos *utterances* como desee el desarrollador de la skill, permitiendo al usuario solicitar la misma acción de muchas maneras distintas, tal y como se puede observar en la Figura 4.6.

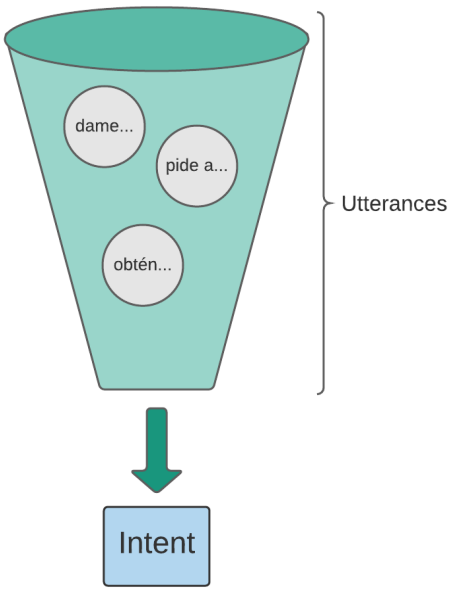


Figura 4.6: Mapeo de *utterances* en un *intent*

Por defecto, una skill recién creada ya dispone de una serie de *intents* denominados *built-in intens*, los cuales son los siguientes:

- *AMAZON.CancelIntent*: responde a palabras y frases que indican que el usuario quiere cancelar la interacción actual. Su aplicación puede utilizar este *intent* para eliminar los valores de tipo *slot* y otros atributos antes de finalizar la interacción con el usuario. Algunos de los *utterances* que se suelen emplear para poder actuar con este *intent* son “cancela” u “olvídalo”.

- *AMAZON.HelpIntent*: responde a las palabras y frases que indican que el usuario necesita ayuda mientras interactúa con el asistente virtual. Cuando se invoca este *intent*, se puede configurar la skill para proporcionar información sobre las capacidades de esta. Los *utterances* más comunes para este *intent* son “ayuda”, “ayúdame” o “necesito ayuda”.

- *AMAZON.StopIntent*: responde a las palabras y frases que indican que el usuario quiere dejar de procesar el *intent* actual y que desea terminar la interacción con esa skill. Algunos de los *utterances* que se suelen usar para este *intent* son “para” o “apaga”. Este *intent* da por terminada la interacción con la skill, a diferencia de *AMAZON.CancelIntent*, que la interacción continúa.

En la definición de uno de estos *intents* ha aparecido el término *slot*. Un *slot* corresponde con una parte del *utterance* que se emplea para poder pasar algún tipo de dato a la skill, como si se tratase de una variable o argumento en algún lenguaje de programación. Estos *slots* están definidos por el tipo de dato que se desea hacer llegar a la skill, habiendo una lista de 17 tipos predefinidos llamados *built-in slot types* (*AMAZON.DATE, AMAZON.DURATION, AMAZON.FirstName, AMAZON.NUMBER, AMAZON.PhoneNumber, AMAZON.TIME…*).

Cuando se definen los distintos *utterances* de cada *intent* solo es necesario escribir el nombre del *slot* entre corchetes, además de definir ese nombre del *slot* junto a su *slot type*. Por ejemplo, en el primer *intent* de la skill implementada en este trabajo, llamado *ObtenerTemperaturaSSIntent*, uno de los *utterances* para poder ejecutar dicho *intent* sería como el de la Figura 4.7.

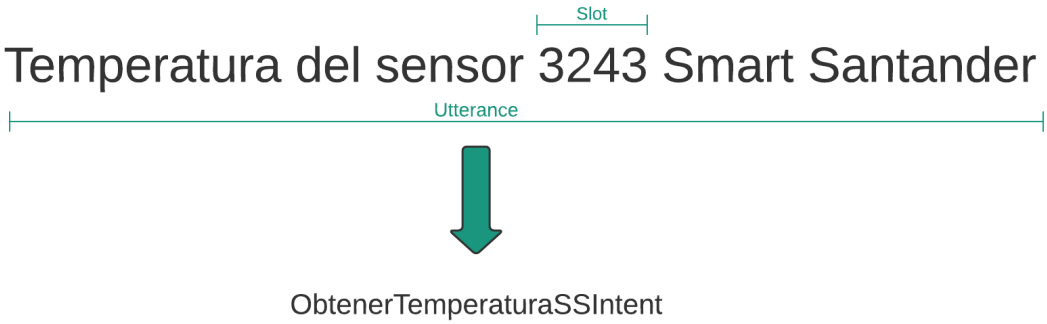


Figura 4.7: Ejemplo de *utterance* con *slot*

En este ejemplo de la Figura 4.7 se puede ver como el *utterance* se traduce en la intención del usuario de conocer el valor de la temperatura de un sensor de la plataforma Smart Santander, y ese sensor es el 3243, cuyo valor se pasa a través de un *slot*. Ese *slot* se llama "numero" y es de tipo *AMAZON.NUMBER*, uno de los *built-in slot types* comentados anteriormente. Este *slot* es el único presente en la skill, y aparece en todos los *intents* de los que esta dispone.

A continuación, se presenta el listado de los *intents* de los que dispone la skill Consulta Sensores.

- *ObtenerTemperaturaSSIntent*: cuando un *utterance* se mapea en este *intent*, lo que realiza la skill es una petición HTTPS con el método GET a la API de Smart Santander, solicitando el valor de la temperatura ambiental del sensor con el número que el usuario haya pasado a través del *slot* "número".

- *ObtenerTemperaturaFWIntent*: cuando un *utterance* se mapea en este *intent*, lo que realiza la skill es una petición HTTP con el método GET a la plataforma de FIWARE, solicitando el valor de la temperatura ambiental del sensor con el número que el usuario haya pasado a través del *slot* "número".

- *ObtenerHumedadSSIntent*: cuando un *utterance* se mapea en este *intent*, lo que realiza la skill es una petición HTTPS con el método GET a la API de Smart Santander, solicitando el valor de la humedad relativa del sensor con el número que el usuario haya pasado a través del *slot* "número".

- *ObtenerHumedadFWIntent*: cuando un *utterance* se mapea en este *intent*, lo que realiza la skill es una petición HTTP con el método GET a la plataforma de FIWARE, solicitando el valor de la humedad relativa del sensor con el número que el usuario haya pasado a través del *slot* "número".

- *ObtenerPosicionSSIntent*: cuando un *utterance* se mapea en este *intent*, lo que realiza la skill es una petición HTTPS con el método GET a la API de Smart Santander, solicitando la posición del sensor con el número que el usuario haya pasado a través del *slot* "número". Esta posición viene dada por las coordenadas en grados decimales.

- *ObtenerPosicionFWIntent*: cuando un *utterance* se mapea en este *intent*, lo que realiza la skill es una petición HTTP con el método GET a la plataforma de FIWARE, solicitando la posición del sensor con el número que el usuario haya pasado a través del *slot* "número". Esta posición viene dada por las coordenadas en grados decimales.

La creación y modificación del modelo de voz se puede llevar a cabo desde el ADC, en la pestaña *Build*. Ahí se encuentra una herramienta llamada *JSON Editor*, la cual traduce todo lo que se realiza de forma gráfica a un archivo JSON. Para desarrollar el modelo de voz de forma local, se puede modelar este *interaction model* modificando ese archivo JSON, en el que vienen definidos el *invocation name* y los distintos *intents* con los *utterances* que mapean en ese *intent* y los *slots* que se requieren.

```
{
    "name": "ObtenerTemperaturaSSIntent",
    "slots": [
        {
            "name": "numero",
            "type": "AMAZON.NUMBER"
        }
    ],
    "samples": [
        "temperatura del sensor {numero} Smart Santander",
        "dame la temperatura del sensor {numero} Smart Santander",
        "dame la temperatura del {numero} Smart Santander",
        "dime la temperatura del sensor {numero} Smart Santander",
        "dime la temperatura del {numero} Smart Santander",
        "temperatura del {numero} Smart Santander",
        "{numero} temperatura Smart Santander"
    ]
},
```

Figura 4.8: *ObtenerTemperaturaIntent* en el modelo de voz

En el ejemplo de la Figura 4.8 se puede ver una entrada de este modelo de voz en el archivo JSON. Cada entrada consta de tres campos, que corresponden con el nombre del *intent*, las definiciones de los *slots* que tiene dicho *intent* y los distintos *utterances* que indican dicha intención. Se puede observar como cada *utterance*, o *sample utterance*, corresponde con una forma distinta de expresar la misma intención. Si el usuario que ejecuta la skill emplea cualquiera de esas siete formulaciones la skill lo traduciría en el *intent* de obtener la temperatura del sensor deseado a través de la API de Smart Santander. Se puede observar como entre corchetes aparece la palabra "numero", que corresponde al nombre del *slot* de tipo *AMAZON.NUMBER* que la skill debe obtener del usuario.

Los demás *intents* tienen una forma muy similar de comportarse al de la Figura 4.8, cambiando ciertas palabras como “temperatura” por “posición” o “humedad”, o también “Smart Santander” por “FIWARE”, para elegir la fuente de la que obtener la información deseada.

4.1.1.3 Endpoint

La forma en que el asistente virtual envía la información captada mediante la interacción con el usuario hacia el back-end de la skill es en formato JSON. Este back-end se encuentra alojado en el *endpoint* de la skill, que por defecto se almacenará en AWS Lambda (apartado 4.2). La skill realiza las acciones necesarias para procesar esa información y devuelve una respuesta en formato JSON, que el asistente virtual procesará para devolver al usuario mediante voz.

Esto se traduce en que el *endpoint* de una skill es el sitio que recibirá un POST cada vez que alguien quiera interactuar con esa skill, y en este pestaña dentro del *Build* es donde el desarrollador debe elegir si el *endpoint* de la skill se alojará en AWS Lambda o en un servicio HTTPS gestionado por el propio desarrollador.

Inicialmente y la opción por defecto es la de AWS Lambda, pero en el apartado 4.3 se puede ver que para alojar la skill sin depender de Amazon es necesario hacer uso de la segunda opción comentada en el párrafo anterior.

4.1.2 Code

En la ventana *Code* del Alexa Developer Console, se proporciona al desarrollador una herramienta de edición de código para poder crear e implementar las distintas funcionalidades que se desee que tenga la skill. Todo el back-end de la skill se puede modificar desde esta herramienta que muestra el código que se encuentra alojado en la nube de Amazon, en caso de tratarse de una *Alexa-hosted skill*. En este tipo de skills, el back-end de dicha skill se almacena usando AWS Lambda, un servicio de Amazon Web Services. Esto se encuentra explicado en el apartado 4.2 de este trabajo.

Este tipo de skill no es el caso de Consulta Sensores, la skill desarrollada en este trabajo, así que la parte de *Code* no ha sido utilizada. El código ha sido almacenado y desarrollado en local, y se puede ver su análisis y explicación en el tercer apartado de este capítulo 4.

4.1.3 Test

Algo de vital importancia la hora de llevar a cabo cualquier tipo de proyecto como desarrollador de software es la importancia de poder probar los avances realizados antes de añadir nuevas funcionalidades. En el caso de cometer algún tipo de error de concepto o de código, siempre es más fácil encontrarlo en un pequeño módulo del código que no en todo ello.

En esta ventana dentro de Alexa Developer Console permite al desarrollador probar la skill que está creando, interactuando con el micrófono o escribiendo por teclado lo que desea decirle al asistente. Las respuestas de la skill son devueltas a través de los altavoces con la voz de Alexa, tal y como lo haría la skill final, y también se muestra el texto por pantalla.

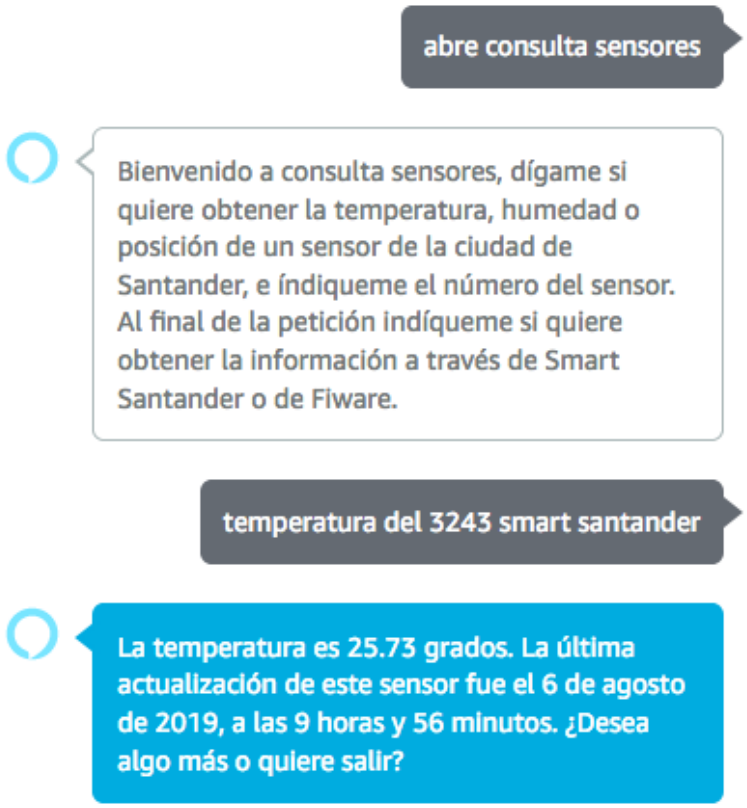


Figura 4.9: Prueba en ADC

En la Figura 4.9 se puede observar una prueba dentro de la ventana *Test* en Alexa Developer Console. Lo que aparece desde la derecha del diálogo es lo que diría un supuesto usuario y lo de la izquierda es lo que responde la skill de Alexa. En este ejemplo se ha lanzado la skill y luego se ha pedido la temperatura de un sensor a través de la API de Smart Santander.

En el resto de la ventana, aparte de lo que se muestra en la Figura 4.9, se puede observar lo que hay en la Figura 4.10. Esto que se ve es el JSON de la petición entrante, que proviene del servicio de Alexa. También se muestra el JSON que genera el back-end de la skill.

También se puede observar en la parte superior una opción que deja marcar el *Device Display*. Esta opción permite visualizar como se vería la interacción con la skill en los dispositivos Echo que disponen de pantalla, como pueden ser el Echo Show 5, el Echo Show 8 o el nuevo Echo Show de 2ª generación. Para poder visualizar una skill por pantalla, el desarrollador de la skill debe emplear Alexa Presentation Language (APL), lo cual es un nuevo lenguaje de diseño y herramientas para enriquecer la experiencia de la skills en los dispositivos con pantalla. En este trabajo no se ha desarrollado este aspecto para la skill Consulta Sensores, pensando únicamente en la experiencia del usuario a través del audio.

Aún así, las skills que no implementan este aspecto funcionan del mismo en todos los dispositivos, tanto en los que no tienen pantalla como en los que sí que disponen de ella.

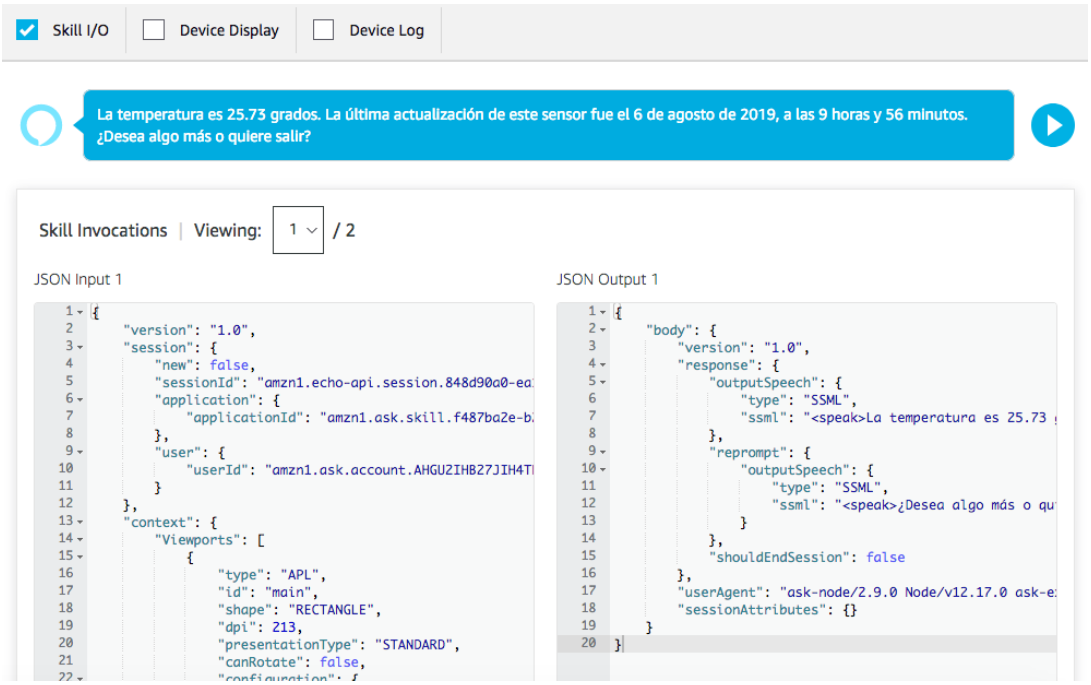


Figura 4.10: Resto de la ventana Test

4.2 AWS Lambda

AWS Lambda [20] en un servicio en la nube de Amazon que permite ejecutar código sin la obligación de poseer y administrar servidores. Este servicio ejecuta el código solo cuando es necesario, y realiza un escalado automático, desde unas peticiones al día hasta millones por segundo. Sólo se paga por el tiempo de cálculo consumido. Con AWS Lambda se puede ejecutar el código de prácticamente cualquier tipo de aplicación o servicio, sin necesidad de administrar nada. AWS Lambda ejecuta el código en una infraestructura informática y realiza toda la administración de los recursos necesarios, incluyendo el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, la supervisión y el registro del código.

Estas funciones Lambda se usan para construir nuevas skills de Alexa, y es la manera que viene por defecto en el momento de crear una skill para alojar todo el código de esta. Es la forma más sencilla de construir una skill, alojándola en la nube de Amazon, ya que el desarrollador se encarga de codificar, pero no de gestionar la ejecución de este. El entorno AWS Lambda se encarga de ejecutarlo en respuesta a las interacciones de voz de Alexa y administra los recursos de computación.

4.2.1 Alojar una skill como función AWS Lambda

Usar AWS Lambda para alojar la skill elimina parte de la complejidad que rodea el desarrollo de un servicio de este tipo. Algunas de las ventajas que proporciona son las siguientes:

- El desarrollador no necesita administrar o gestionar ninguno de los recursos de computación necesarios para el servicio.
- No necesita un certificado SSL.
- No necesita verificar que las solicitudes provienen del servicio de Alexa. El acceso para ejecutar el back-end de la skill está controlado por permisos dentro de AWS.
- AWS Lambda ejecuta el código solo cuando es necesario y se encarga del escalado de los recursos, por lo que no es necesario aprovisionar o ejecutar continuamente los servidores.
- Alexa encripta las comunicaciones con AWS Lambda utilizando TLS.
- Para la mayoría de los desarrolladores es suficiente con los recursos gratuitos que proporciona Lambda para alojar una skill de Alexa. Se dispone de un millón de solicitudes gratuitas al mes. Esta plan está disponible indefinidamente, y no expira.

AWS Lambda soporta varios lenguajes, incluyendo Node.js, que es el que se emplea en este trabajo, Python y Java.

Lo que hace la función AWS Lambda es manejar las peticiones que recibe desde Alexa, encargándose también de las solicitudes y respuestas en formato JSON.

En el desarrollo de este trabajo se ha preferido la opción de alojar el código de la skill como un servicio web, para dotar al proyecto de independencia de Amazon, y llegado al caso de poder reutilizar parte de las funcionalidades para el acceso a través de otros asistentes virtuales. De este modo, como desarrollador, me encargo prácticamente de todo lo relacionado con la skill.

4.3 Alternativa a AWS Lambda

En este trabajo se ha decidido no usar el servicio de Amazon para alojar la skill, sino que se ha creado un servidor propio en el que alojar el back-end de la skill. Este servidor recibirá las peticiones del servicio de Alexa para procesarlas y devolver una respuesta.

Para poder implementar esta alternativa, además de crear el servicio web, es necesario realizar una serie de modificaciones del código que habría en el caso de tratarse de una *Alexa Hosted-skill*. También serán necesarias modificaciones en los

ajustes del *endpoint* de la skill, a través del ADC. Estas modificaciones y el desarrollo completo de la skill pueden verse desarrollado en el capítulo siguiente.

5 Alexa Skill: desarrollo

En este capítulo se va a profundizar en el funcionamiento del sistema diseñando, ampliando el funcionamiento visto en la Figura 4.1. En este capítulo aparece un elemento nuevo, que es el servicio web que aloja la skill, y que introduce un cambio significativo en la forma de operar del sistema con respecto a lo visto en capítulos anteriores. Para entender este proceso, es necesario analizar con detenimiento la Figura 5.1.

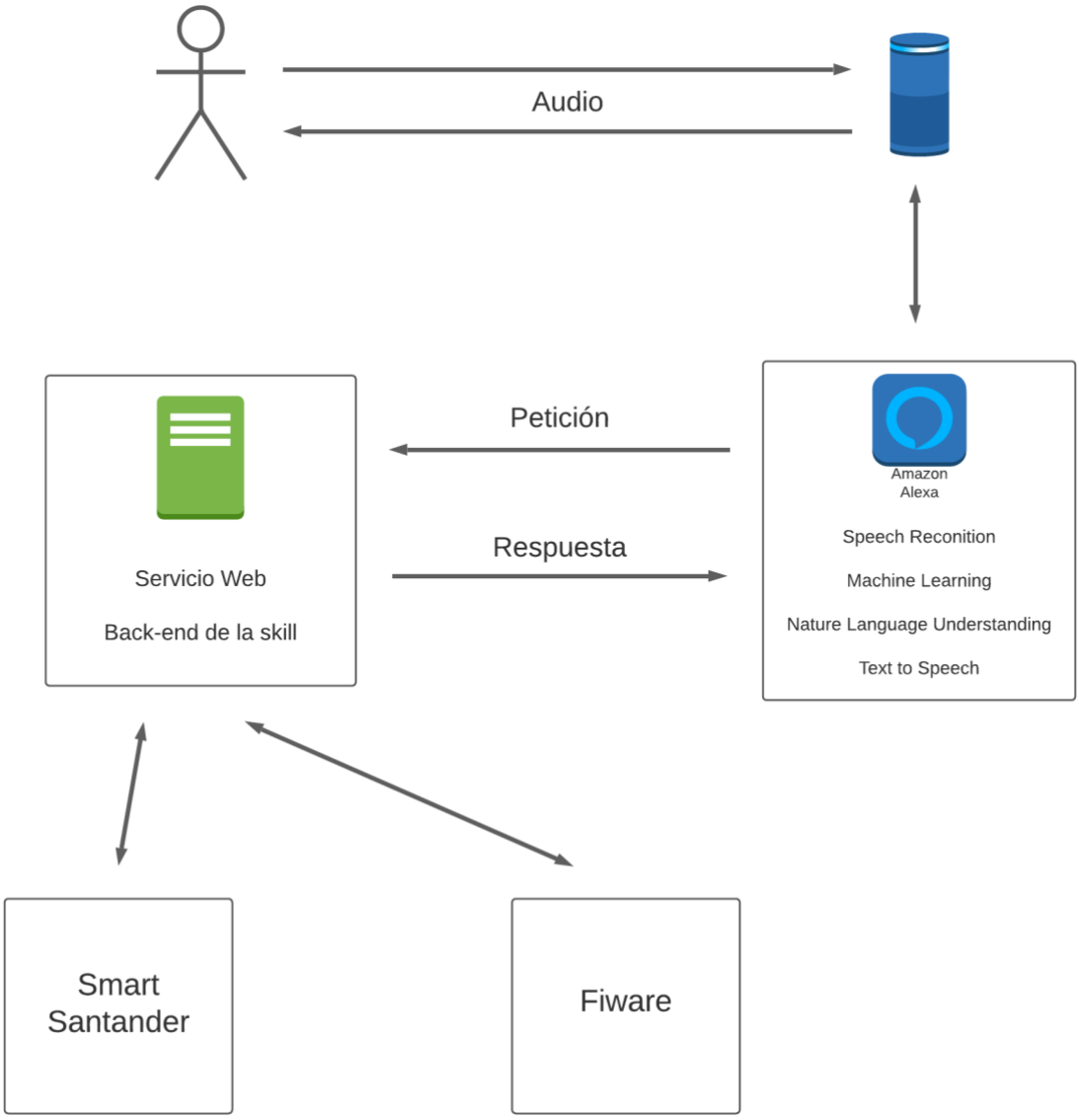


Figura 5.1: Desarrollo completo del procesamiento de peticiones

En la Figura 5.1 se pueden apreciar los distintos actores que intervienen en todo el desarrollo de recogida de datos, procesamiento de esos datos y presentación de resultados al usuario. Por orden de arriba a abajo, en primer lugar, aparece el usuario. Este usuario es quien da instrucciones y realiza peticiones al asistente virtual para satisfacer sus necesidades.

En segundo lugar, aparece el dispositivo con el que interactúa. Este dispositivo puede ser uno de los ya comentados de la clase Echo o un dispositivo móvil, el cual puede hacer uso de la app de Alexa para poder hacer uso de las skills. Este dispositivo actúa como el interfaz con el cliente. Lo que hace es recoger la información de la skill con la que el usuario desea interactuar y recopilar las órdenes que este realiza. Una vez conocida la skill, el dispositivo pasa la orden al servicio en la nube de Alexa, que es el encargado de la parte inteligente del asistente virtual.

Esta parte inteligente que se comenta es la que se explica en el capítulo 4. El servicio en la nube de Alexa realiza una etapa de *Automated Speech Recognition*, en la que transforma el audio a texto, y otra etapa de *Natural Language Understanding*, en el que dota de sentido a ese texto, para posteriormente enviar la información necesaria al back-end de la skill alojado como servicio web propio.

En el servicio web se encuentra el back-end de la skill explicado en el apartado 4.3 de este trabajo. Cuando le llegan las órdenes traducidas un *intent* y los valores de los *slots* correspondientes, este servicio realiza las peticiones a la API de Smart Santander o la plataforma FIWARE, según corresponda, mediante el método GET del protocolo HTTP.

Estas dos plataformas lo único que hacen es recibir la petición y buscar dicha información en su base de datos. En esta base de datos se encontrará el valor del sensor solicitado más reciente. A continuación, devuelven al servicio web la información solicitada en un archivo con formato JSON.

El camino de vuelta para proporcionar la información al usuario es muy similar. El servicio web recoge la información proporcionada por la plataforma correspondiente y realiza las operaciones necesarias antes de proporcionar a Alexa la respuesta que debe dar al usuario, en formato de texto. Lo siguiente será que Alexa transforme ese texto en audio, para que el interfaz de entrada – salida reproduzca la respuesta y le pueda llegar al usuario.

5.1 Alexa en servidor propio

La otra alternativa a lo comentado en el capítulo anterior implica crear una skill personalizada implementando un servicio web que acepte solicitudes y envíe respuestas al servicio de Alexa en la nube. Para poder hacerlo de este manera, dicho servicio web debe cumplir una serie de requisitos.

- El servicio debe ser accesible a través de Internet.
- El servicio debe aceptar peticiones HTTP en el puerto 443.
- El servicio debe soportar HTTP sobre SSL/TLS, usando un certificado de confianza de Amazon. El nombre del dominio del servicio web debe aparecer en la sección *Subject Alternative Name* (SAN) del certificado.

- El servicio debe verificar que las solicitudes que le llegan provienen de Alexa.

- El servicio debe adherirse a la interfaz de Alexa Skills Kit.

El tercer punto es importante porque cuando Alexa se comunica con el servicio web en el que se encuentra la skill, las solicitudes de los usuarios y las respuestas correspondientes se transmiten por Internet, por lo que para proteger la confidencialidad e integridad de estos datos es necesario que se cumplan estrictamente que las conexiones HTTP sean seguras, para lo que utiliza SSL/TLS. Esto significa que dicho servicio web debe presentar un certificado válido y de confianza cuando se establece la conexión.

Para ello, es necesario especificar el tipo de certificado SSL que posee el servicio web, y esto se hace en la consola de desarrollo del Alexa Skills Kit, es decir, en el Alexa Developer Console. En la ventana *Build*, dentro de la sección *Endpoint*, se puede cambiar que la skill esté alojada como función AWS Lambda y elegir que se aloje como un servicio web manejado por el desarrollador. Una vez ahí se puede elegir el tipo de certificado, teniendo tres opciones.

La primera de las opciones del certificado del servicio web es la de un certificado firmado por una autoridad de confianza de Amazon, en cuyo caso habrá que elegir la opción “My development endpoint has a certificate from a trusted certificate authority”. La segunda opción corresponde a la utilización de un certificado comodín, es decir, un certificado que proporciona SSL para varios subdominios. En cualquier caso, este certificado deberá estar firmado por una autoridad de certificación de confianza de Amazon. En esta opción habrá que elegir donde indica “My development endpoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority”. La última de las opciones dice “I will upload a self-signed certificate in X.509 format”, que sería la opción elegida para subir un certificado autofirmado a la consola de desarrollo de Alexa (ADC) y configurar el servicio web para que presente ese certificado. Esta opción se suele utilizar para las fase de pruebas. En este trabajo se utiliza la segunda de estas opciones, como será explicado más adelante.

En el cuarto punto de la lista de requerimientos para alojar la skill como un servicio web se enuncia que el servicio web debe verificar que las peticiones entrantes provienen de Alexa. Esto es necesario porque las solicitudes que el Alexa Voice Service envía al servicio web que aloja la skill se transmiten por Internet, y con el objetivo de proteger dicho servicio web de potenciales atacantes, es necesario verificar que dichas solicitudes son enviadas por ese servicio de Alexa. Además, las skills que no implementen esta función no podrán ser certificadas por Amazon y no se podrán ofrecer a los usuarios finales. La forma en la que se ha implementado esta funcionalidad en la skill desarrollada en este proyecto puede ser vista en el apartado 5.1.1.2.

5.1.1.1 Ngrok y el endpoint

Para poder desarrollar la skill como un servicio web, se necesita modificar la opción elegida en el *endpoint*. De las tres opciones disponibles, comentadas en el apartado 5.1 unos párrafos más arriba, la elegida será la que dice “My development

endoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority" (Figura 5.2). Para obtener un dominio en el que ejecutar el servicio web se emplea la herramienta Ngrok. [21]

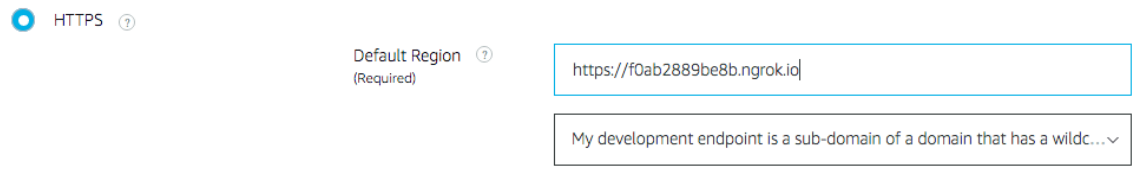


Figura 5.2: Configuración *endpoint*

El servicio web en el que se aloja la skill se ejecuta en local, en el ordenador en el que se desarrolla la skill, en el puerto 8080. Ngrok permite obtener un dominio externo que se vincula al servicio corriendo en local en el puerto que se desee, es decir, lo que hace es abrir el servidor local a Internet. Lo único que hay que hacer es descargar el programa de Ngrok, el cual apenas tiene 3.8MB de tamaño, y ejecutar un comando. En ese momento el servicio web ya dispone de una URL obtenida dinámicamente, y será válida mientras Ngrok esté corriendo. En la Figura 5.3 se puede ver la interfaz de Ngrok al ejecutar el comando para comenzar a correr el programa.

Figura 5.3: Interfaz de Ngrok

5.1.1.2 Servidor con Express/Node

El servicio web se crea con Express [22], un framework web escrito en JavaScript y alojado dentro de entorno de ejecución Node JS. En este apartado se van a analizar los conceptos clave para comprender el servicio que se ha desarrollado en este proyecto.

Node JS es un entorno de código abierto que permite a desarrolladores crear toda clase de herramientas de lado servidor y aplicaciones en JavaScript. Son numerosas las ventajas que proporciona este entorno, como un rendimiento optimizado, gran escalabilidad, las ventajas que proporciona JavaScript, gran compatibilidad de sistemas operativos y una comunidad de desarrolladores muy activa.

Express es un framework de Node JS que proporciona mecanismos para la escritura de manejadores de peticiones HTTP, para establecer ajustes de aplicaciones web como qué puerto usar para conectar y para añadir procesamiento de peticiones a través de *middleware*, entre otras ventajas.

En la Figura 5.4 se puede ver el código del servidor web. Las líneas 1 y 4 corresponden con la inclusión del módulo de Express y la creación de la aplicación. Este segundo elemento posee métodos para el enrutamiento de peticiones HTTP. La línea 2 incluye el adaptador `ask-sdk-express`, mediante el cual se proporciona el código necesario para verificar las peticiones y la verificación de fecha y hora, algo que debe tener la skill para poder alojarla como un servicio web.

En las líneas de la 5 a la 7 se construye la skill sobre el servicio web. Esta skill se encuentra en el archivo `skillBuilder.js`, como se puede ver en la línea 5. En la línea 9 se define la ruta del servicio web, y como el adaptador, a través de la función `getRequestHandlers` recoge las peticiones a la skill para poder se procesadas por el back-end de la skill. Las tres últimas líneas del código definen el puerto en el que escucha el servicio web, que será el 8080. Se puede observar como este puerto coincide con elegido para configurar Ngrok en la Figura 5.3.

```
1 const express = require('express');
2 const { ExpressAdapter } = require('ask-sdk-express-adapter');
3
4 const app = express();
5 const skillBuilder = require('./skillBuilder.js').skillBuilder;
6 const skill = skillBuilder.create();
7 const adapter = new ExpressAdapter(skill, true, true);
8
9 app.post('/', adapter.getRequestHandlers());
10 app.listen((8080), () => {
11     console.log(`Server on port ${8080}`);
12 });
```

Figura 5.4: Código del servidor web

5.1.1.3 SkillBuilder

En el archivo `skillBuilder.js` se encuentra el código de la skill. En este código se encuentran definidos los *handlers*, que son las porciones de código que se ejecutan dependiendo del *intent* que corresponda ejecutar.

```
1 const Alexa = require('ask-sdk-core');
2 const logic = require('./logic');
3 const moment = require('moment');
```

Figura 5.5: Módulos importados en `skillBuilder.js`

En la Figura 5.5 se ven los módulos empleados en esta parte del código, que corresponden con el core de Alexa, la inclusión del archivo `logic.js` y el módulo `Moment` [23]. El archivo `logic.js` aloja las funciones necesarias para realizar las peticiones HTTP a la API de Smart Santander y a la plataforma FIWARE. El módulo `Moment` se emplea para realizar operaciones relacionadas con los formatos

de las fechas obtenidos en las respuestas de API Smart Santander y la plataforma FIWARE.

Ha sido mencionado que un *handler* es una porción de código que se ejecuta dependiendo del *intent* que haya sido solicitado por el usuario. Por lo tanto, se confirma que `skillBuilder.js` consta de un *handler* por cada *intent* implementado por la skill. Además de los *handlers* correspondientes a los *intents* creados que se enumeraron en el apartado 4.1.1.2, hay un *handler* para el lanzamiento de la skill, otro para el *built-in intent* de ayuda, otro para el de cancelar y parar y otro para terminar la sesión.

La arquitectura de un *handler* es la que se puede observar en la Figura 5.6. El *handler* tiene el nombre del *intent* con el que está relacionado, y consta de dos partes claramente diferenciadas.

- *canHandle*: esta parte es la encargada de determina qué tipo de evento está sucediendo, cuál es el tipo de petición entrante. Los tipos que aparecen en la skill son `LaunchRequest` para el *handler* que maneja en lanzamiento de la aplicación e `IntentRequest` para el resto.

- *handle*: esta parte del *handler* solo se ejecutará en el caso de que el *return* del `canHandle` devuelva un true. Después de las operaciones necesarias que realiza esta parte del *handler*, aparece un *return*, que es donde se da la respuesta al *Alexa Voice Service* para que se la transmita al usuario. En este *return*, se devuelve el texto con el contenido que queremos que reproduzca Alexa. La variable `speakOutput` es una cadena de caracteres que será definida dentro del *handle*.

```
const NombreIntentHandler = {
    canHandle(handlerInput) {
        return handlerInput.requestEnvelope.request.type === 'IntentRequest'
            && handlerInput.requestEnvelope.request.intent.name === 'NombreIntent';
    },
    handle(handlerInput) {
        /* Código que ejecuta el intent */

        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput)
            .getResponse();
    }
};
```

Figura 5.6: Arquitectura de un *handle*

Para comprender mejor este funcionamiento será de utilidad observar la Figura 5.7 y la Figura 5.8.

La Figura 5.7 corresponde al *handler* que lanza la skill. Se puede ver como el requisito para ejecutar el código es que el tipo de petición sea `LaunchRequest`, es decir, de querer comenzar a interactuar con la skill. En ese caso se ejectua el *handle*, en el que se define el texto que pronunciara la skill Consulta Sensores para dar la bienvenida al usuario.

```
const LaunchRequestHandler = {
    canHandle(handlerInput) {
        return handlerInput.requestEnvelope.request.type === 'LaunchRequest';
    },
    handle(handlerInput) {
        const speakOutput = 'Bienvenido a consulta sensores, dígame si quiere obtener la
        temperatura, humedad o posición de un sensor de la ciudad de Santander, e indíqueme
        el número del sensor. Al final de la petición indíqueme si quiere obtener la
        información a través de <lang xml:lang="en-US">Smart</lang> Santander o de <lang
        xml:lang="en-US">Fiware</lang>.';

        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput)
            .getResponse();
    }
};
```

Figura 5.7: `LaunchRequestHandler`

```
const ObtenerTemperaturaSSIntentHandler = {
    canHandle(handlerInput) {
        return handlerInput.requestEnvelope.request.type === 'IntentRequest'
            && handlerInput.requestEnvelope.request.intent.name ===
            'ObtenerTemperaturaSSIntent';
    },
    async handle(handlerInput) {
        const intent = handlerInput.requestEnvelope.request.intent;
        var mes = ["enero", "febrero", "marzo", "abril", "mayo", "junio",  "julio",
        "agosto", "septiembre", "octubre", "noviembre", "diciembre"];

        const numero = intent.slots.numero.value;
        const phenomenon = 'temperature:ambient';
        const response = await logic.httpGetSmartSantader(numero,phenomenon);
        console.log(response);

        const temperatura = response.value;
        var momentDate = moment(response.timestamp);
        var dia = momentDate.date();
        var month = momentDate.month();
        var year = momentDate.year();
        var hora = momentDate.hours()-2;
        if (hora<0) {
            hora = hora + 24;
            dia = dia - 1;
            if (dia==0) {
                month = month - 1;
                if (month==0) {
                    year = year - 1;
                }
            }
        }
        var minuto = momentDate.minutes();

        let speechText = `La temperatura es ${temperatura} grados. La última actualización
        de este sensor fue el ${dia} de ${mes[month]} de ${year}, a las ${hora} horas y
        ${minuto} minutos.`;
        let helpmsg = ' ¿Desea algo más o quiere salir?'
        speechText += ' ' + helpmsg;

        return handlerInput.responseBuilder
            .speak(speechText)
            .reprompt(helpmsg)
            .getResponse();
    }
};
```

Figura 5.8: `ObtenerTemperaturaSSIntentHandler`

En la Figura 5.8 se ve la parte de código correspondiente al *handler* que lleva a cabo la intención de obtener la temperatura de un sensor a través de la API de Smart Santander. En el `canHandle` se aprecia como los requisitos para ejecutar el *handle* es que la petición sea de tipo `IntentRequest` y que corresponda al *intent* llamado `ObtenerTemperaturaSSIntent`. En la línea 10 de la Figura 5.8 se encuentra la variable que recoge la información pasada por el usuario en el slot "número". Ese valor se usará como argumento en la llamada a la función `httpGetSmartSantander`, junto con el fenómeno físico a medir, para realizar la solicitación de la información a la API de Smart Santander. Lo que permite hacer esta petición se verá en el apartado 5.1.1.4.

A partir de ahí, en la variable "response" se tiene la respuesta de la solicitud a la información de dicho sensor. En las siguientes líneas del código se obtiene dicho valor y se realizan las operaciones necesarias para proporcionar la última fecha de actualización del sensor. Por último, en las líneas 33 a 35, se crea la respuesta que ofrecerá Alexa al usuario.

El *handler* de la Figura 5.9 corresponde con aquel que lleva a cabo los *intents* para obtener la temperatura de un sensor a través de FIWARE. El código es prácticamente idéntico al de la Figura 5.8, cambiando la función mediante la cual se hace la petición y los parámetros que está lleva consigo.

Estos dos *handlers* se completan con otros cuatro, que son los encargados de solicitar la humedad y la posición a través de ambas plataformas. El funcionamiento de estos *handlers* es muy similar a los de la Figura 5.8 y la Figura 5.9. Las diferencias con estos se encuentran en el tipo de fenómeno solicitado y en el tipo del procesamiento de los datos de la respuesta a la petición de la plataforma correspondiente. Para implementar la funcionalidad de solicitar algún otro fenómeno físico el esquema del *handler* también sería como el de estas figuras anteriormente comentadas.

Estos seis *handlers* recién comentados forman la parte más funcional de la skill, la que realiza peticiones externas tanto a Smart Santander como a FIWARE. El resto de *handlers* apenas presentan un breve diálogo para ser reproducido por Alexa, sin ninguna operación compleja, como es el caso del `HelpIntentHandler` que se puede ver en la Figura 5.10. Cuando el usuario solicita ayuda se reproduce un mensaje con las indicaciones necesarias para que el usuario se familiarice con la skill y conozca la formulación correcta de las peticiones que debe realizar.

Al final del código que posee el archivo `skillBuilder.js`, aparece el código que se puede ver en la Figura 5.11. Si bien lo anteriormente explicado es la propia implementación de los métodos que soporta la skill, el código que se muestra en este caso incluye la definición y construcción de la skill que se va a exportar. Si alguno de los *handlers* que aparecen anteriormente no se incluyen en el listado de `RequestHandlers` estos no tendrían una referencia y por tanto no se ejecutarían.

Adicionalmente, para el caso en el que se deseare que la skill se ejecutara con función de AWS Lambda se indicaría añadiendo la solicitud de convertirla o tratarla como lambda a través de `.lambda()`, como se puede ver en la Figura 5.12.

```
const ObtenerTemperaturaFWIntentHandler = {
    canHandle(handlerInput) {
        return handlerInput.requestEnvelope.request.type === 'IntentRequest'
            && handlerInput.requestEnvelope.request.intent.name ===
'ObtenerTemperaturaFWIntent';
    },
    async handle(handlerInput) {
        var mes = ["enero", "febrero", "marzo", "abril", "mayo", "junio", "julio",
        "agosto", "septiembre"," octubre", "noviembre", "diciembre"];

        const numero = intent.slots.numero.value;
        const phenomenon = 'temperature';
        const path = 'environment:fixed';
        const response = await logic.httpGetFiware(numero,phenomenon,path);
        console.log(response);

        const temperatura = response.temperature;
        var momentDate = moment(response.dateObserved);
        var dia = momentDate.date();
        var month = momentDate.month();
        var year = momentDate.year();
        var hora = momentDate.hours()-2;
        if (hora<0) {
            hora = hora + 24;
            dia = dia - 1;
            if (dia==0) {
                month = month - 1;
                if (month==0) {
                    year = year - 1;
                }
            }
        }
        var minuto = momentDate.minutes();

        let speechText = `La temperatura es ${temperatura} grados. La última actualización
        de este sensor fue el ${dia} de ${mes[month]} de ${year}, a las ${hora} horas y
        ${minuto} minutos.`;
        let helpmsg = ' ¿Desea algo más o quiere salir?'
        speechText += ' ' + helpmsg;

        return handlerInput.responseBuilder
            .speak(speechText)
            .reprompt(helpmsg)
            .getResponse();
    }
};
```

Figura 5.9: `ObtenerTemperaturaFWIntentHandler`

```
const HelpIntentHandler = {
    canHandle(handlerInput) {
        return handlerInput.requestEnvelope.request.type === 'IntentRequest'
            && handlerInput.requestEnvelope.request.intent.name === 'AMAZON.HelpIntent';
    },
    handle(handlerInput) {
        const speakOutput = 'Puedes decirme el número de un sensor para obtener su
        temperatura, su humedad o las coordenadas en las que se encuentra. Además, debes
        indicar al final de la petición la plataforma de la que deseas obtener los datos,
        bien <lang xml:lang="en-US">Smart</lang> Santander o <lang xml:lang="en-
        US">Fiware</lang>. ¿En qué puedo ayudarte?';

        return handlerInput.responseBuilder
            .speak(speakOutput)
            .reprompt(speakOutput)
            .getResponse();
    }
};
```

Figura 5.10: `HelpIntentHandler`

```
// The SkillBuilder acts as the entry point for your skill, routing
all request and response payloads to the handlers above. Make sure
any new handlers or interceptors you've defined are included below.
The order matters - they're processed top to bottom.
exports.skillBuilder = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        ObtenerTemperaturaSSIntentHandler,
        ObtenerTemperaturaFWIntentHandler,
        ObtenerHumedadSSIntentHandler,
        ObtenerHumedadFWIntentHandler,
        ObtenerPosicionSSIntentHandler,
        ObtenerPosicionFWIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        SessionEndedRequestHandler,
        IntentReflectorHandler) // make sure IntentReflectorHandler
        is last so it doesn't override your custom intent handlers
    .addErrorHandlers(
        ErrorHandler)
    .addRequestInterceptors(
        //interceptors.LocalisationRequestInterceptor,
        LoggingRequestInterceptor)
        //interceptors.LoadAttributesRequestInterceptor
    .addResponseInterceptors(
        LoggingResponseInterceptor)
        //interceptors.SaveAttributesResponseInterceptor
```

Figura 5.11: Exportación de *handlers*

```
// The SkillBuilder acts as the entry point for your skill, routing all request and
response
// payloads to the handlers above. Make sure any new handlers or interceptors you've
// defined are included below. The order matters - they're processed top to bottom.
exports.handler = Alexa.SkillBuilders.custom()
    .addRequestHandlers(
        LaunchRequestHandler,
        ObtenerTemperaturaSSIntentHandler,
        ObtenerTemperaturaFWIntentHandler,
        ObtenerHumedadSSIntentHandler,
        ObtenerHumedadFWIntentHandler,
        ObtenerPosicionSSIntentHandler,
        ObtenerPosicionFWIntentHandler,
        HelpIntentHandler,
        CancelAndStopIntentHandler,
        SessionEndedRequestHandler,
        IntentReflectorHandler) // make sure IntentReflectorHandler is last so it
        doesn't override your custom intent handlers
    .addErrorHandlers(
        ErrorHandler)
    .addRequestInterceptors(
        //interceptors.LocalisationRequestInterceptor,
        LoggingRequestInterceptor)
        //interceptors.LoadAttributesRequestInterceptor
    .addResponseInterceptors(
        LoggingResponseInterceptor)
        //interceptors.SaveAttributesResponseInterceptor
    .lambda();
```

Figura 5.12: Exportación de *handlers* con lambda

5.1.1.4 Logic

En el archivo `logic.js`, tercera parte del código JavaScript que forma la skill, se encuentran las funcionalidades lógicas que se realizan en los *handlers*. Con el objetivo de escribir un código más limpio y depurado y dada la repetición de estas funciones, se toma la decisión de extraerlas del código del skillBuilder para importarlas desde un archivo externo llamado `logic.js`, tal como se ve en la Figura 5.5. A continuación, en la Figura 5.13, se puede ver el código que forma esta parte lógica de la skill, formada básicamente por las librerías *http* y *https* empleadas para realizar las peticiones y la implementación de dos funciones que realizan el GET a Smart Santander y a FIWARE.

```
const https = require('https');
const http = require('http')

module.exports = {
    httpGetSmartSantader(numero,phenomenon) {
        return new Promise((resolve, reject) => {
            var options = {
                [redacted]
                method: 'GET',
                rejectUnauthorized: false,
                requestCert: false,
                agent: false
            };

            const request = https.request(options, (response) => {
                response.setEncoding('utf8');
                let returnData = '';

                response.on('data', (chunk) => {
                    returnData += chunk;
                });

                response.on('end', () => {
                    resolve(JSON.parse(returnData));
                });

                response.on('error', (error) => {
                    reject(error);
                });
            });
            request.end();
        });
    },
    httpGetFiware(numero,phenomenon,path) {
        return new Promise((resolve, reject) => {
            var options = {
                [redacted]
                method: 'GET',
                rejectUnauthorized: false,
                requestCert: false,
                agent: false
            };

            const request = http.request(options, (response) => {
                response.setEncoding('utf8');
                let returnData = '';

                response.on('data', (chunk) => {
                    returnData += chunk;
                });

                response.on('end', () => {
                    resolve(JSON.parse(returnData));
                });

                response.on('error', (error) => {
                    reject(error);
                });
            });
            request.end();
        });
    },
}
```

Figura 5.13: `Logic.js`

El esquema de ambas funciones es prácticamente el mismo. Una primera parte con las opciones de la petición, donde se indica el *host*, el puerto y el *path* donde se realiza la petición. Esta parte aparece oculta en la Figura 5.13 por motivos de seguridad. Aquí es donde se diferencian estas funciones, ya que la primera va dirigida a la API de Smart Santander y la segunda a FIWARE. Siguiendo en las opciones de la petición aparece el método HTTP, que es un GET. Seguido de tres opciones necesarias para evitar que se solicite identificación al intentar acceder a la información. El resto de ambas funciones es idéntico, y corresponde a la parte en la que se realiza la petición o *request* y donde se *parsean* los datos solicitados.

5.1.2 Ventajas e inconvenientes de alojar la skill como servicio web

Esta forma de alojar el código back-end de la skill, alejándose de la dependencia de la nube de Amazon presenta ventajas e inconvenientes, y es importante definir estos aspectos a la hora de elegir que método de trabajo se va a emplear.

Ventajas

- Manejo completo de todo el proceso: el desarrollador tiene el control completa de la skill ya que solo necesita hacer uso de los servicios de Alexa para el reconocimiento de voz del usuario. El código es únicamente suyo y las peticiones se remiten al servicio web donde se aloja la skill.

- Seguridad: los datos de la skill sólo son accesibles por su propietario y nadie más puede ver el código ni ningún otro dato relevante.

Inconvenientes

- Escalabilidad: la potencia de procesamiento y el espacio de almacenamiento tienen un límite fijo, mientras que usando los recursos en la nube la escalabilidad es inmediata, ajustando los recursos acorde a lo que se necesita.

- Ejecutar código: para habilitar la skill es necesario poner a correr el servicio web, mientras que en la nube la skill está siempre ejecutándose y se puede hacer uso de la skill siempre que se desee.

- Mayores conocimientos: son necesarios mayores conocimientos de programación para poder desarrollar las mismas funcionalidades que en el caso de alojar la skill en la nube.

- Fiabilidad: el proveedor del servicio en la nube se encarga de que el servicio esté siempre operativo, así como de balanceos de carga. Además, estos servicios cuentan con redundancia ante fallos para asegurar el correcto funcionamiento.

- Precio: en el caso de tener una skill que recibiese un gran numero de peticiones y que necesitase de grandes recursos de cómputo habría que estudiar que opción es más rentable, si disponer de un servidor físico

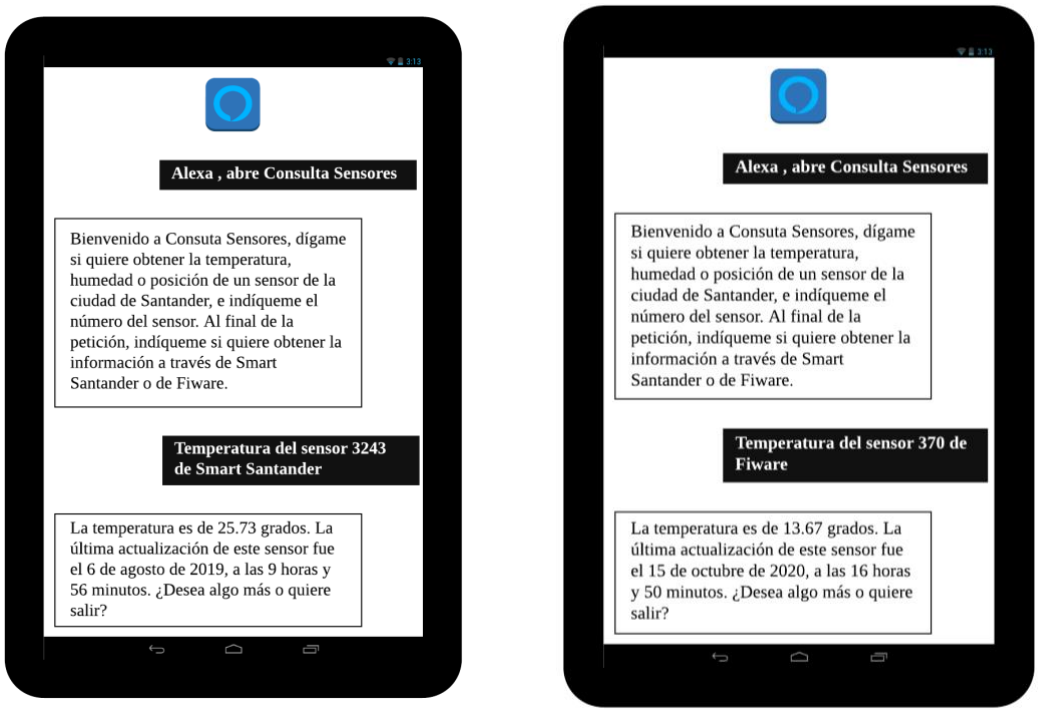
donde correr dicha skill o alojarla en la nube de Amazon. A priori, la opción más rentable es la segunda, porque el método on-premise requiere de la compra del servidor, la instalación, gastos de electricidad, etc.

En definitiva, ambos tipos de alojamiento de la skill tienen sus ventajas e inconvenientes. Es importante conocer el tipo de servicio que se ofrece para elegir la mejor opción. En uno de este tipo, como una skill que requiere pocos recursos de computación es bastante adecuada la opción de alojarla como un servicio web. Si se desease ofrecer la skill al público habría que estudiar la opción de disponer de un servidor que tenga el servicio ejecutándose continuamente para que cualquier usuario pueda hacer uso cuando desee.

5.2 Evaluación de la skill

Se han analizado todos los aspectos relevantes que rodean a la skill "Consulta Sensores", pero aún queda ver como son todas estas funcionalidades que proporciona la herramienta creada. A continuación, se presentan un cúmulo de figuras que representan como es la interacción del usuario con la skill.

En Figura 5.14, Figura 5.15 y Figura 5.16 está representado el diálogo que se mantiene entre el usuario y la skill "Consulta Sensores". Los recuadros con el fondo negro representan lo que dice el usuario y los del fondo blanco lo que responde el asistente virtual. Estos seis modelos de diálogo son todos los posibles con esta skill, a parte de la solicitud de ayuda y de la de despedirse, las cuales se muestran en la Figura 5.17 y Figura 5.18, respectivamente.

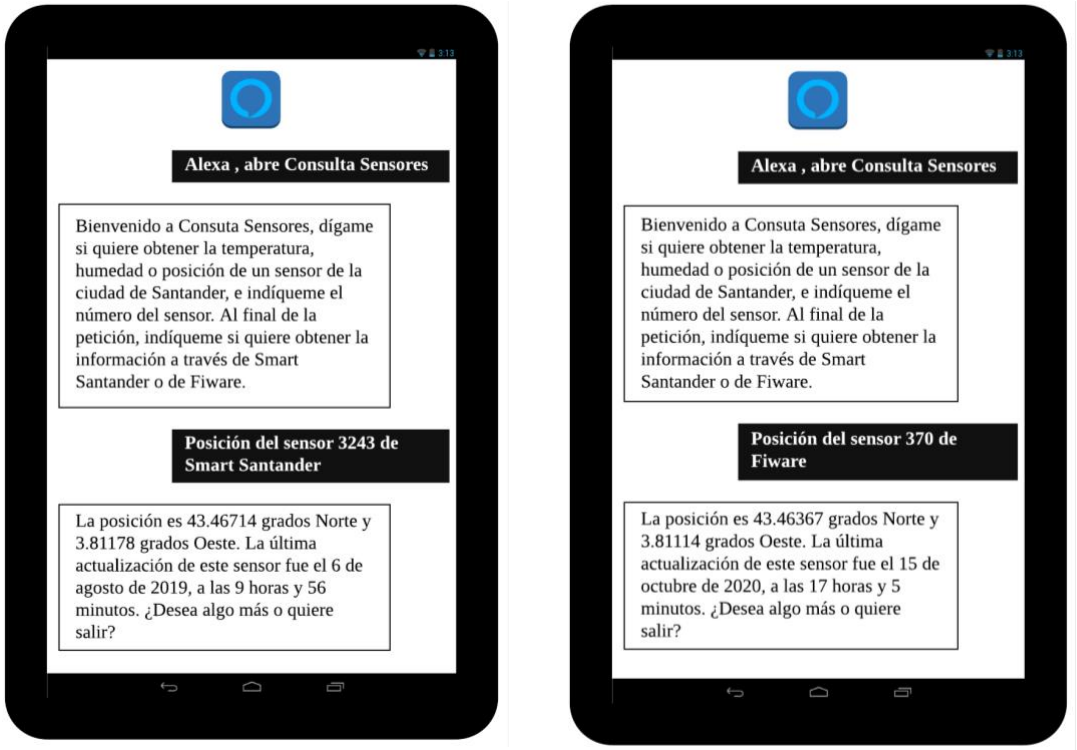


a) Smart Santander b) FIWARE

Figura 5.14: Solicitar temperatura

a) Smart Santander b) FIWARE

Figura 5.15: Solicitar humedad



a) Smart Santander b) FIWARE

Figura 5.16: Solicitar posición

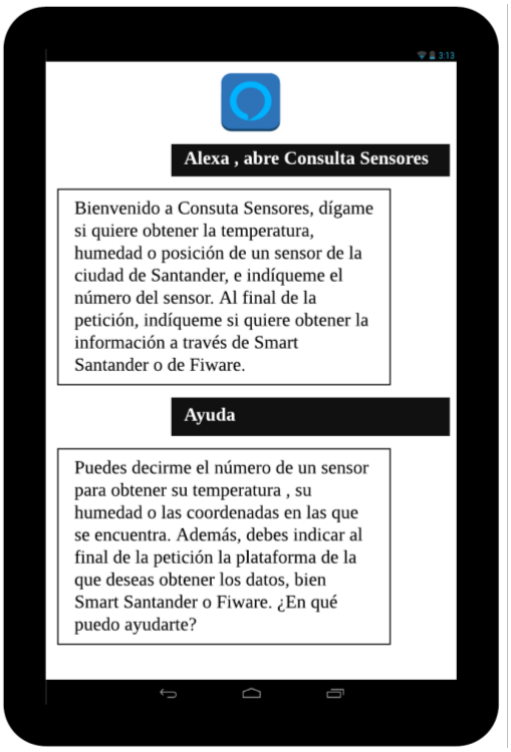


Figura 5.17: Solicitar ayuda

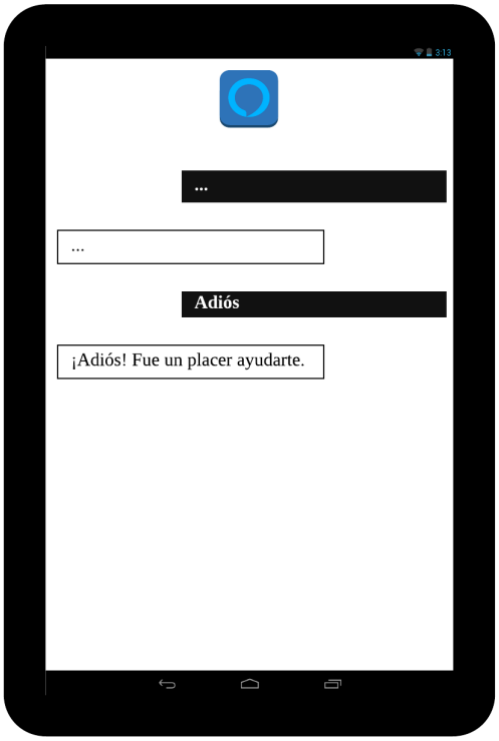


Figura 5.18: Solicitar fin de interacción

5.3 Acceso a la información

En este apartado se puede ver como se realiza el acceso a los datos a través de la API de Smart Santander y de la plataforma FIWARE, tanto la forma de realizar la petición como el formato de las respuestas que ambas plataformas devuelven al servicio web que aloja la skill.

5.3.1 Acceso a datos de Smart Santander

Para acceder a la información de Smart Santander es suficiente con realizar una petición HTTPS mediante el método GET a la dirección correspondiente. En el path de la URL habrá que añadir datos como el tipo de fenómeno físico que se desea medir y el número den sensor del que se desea recabar dicha información.

En el siguiente ejemplo se puede observar la respuesta que ofrece la API de Smart Santander, cuya manera de devolver los datos es en formato JSON.

```
urn:                "urn:x-iot:smartsantander:u7jcfa:t3243"
timestamp:          "2019-08-06T09:56:08.446Z"
location:
    coordinates:
          0:    -3.81178
          1:    43.46714
    type:       "Point"
value:          25.73
phenomenon:     "temperature:ambient"
uom:            "degreeCelsius"
```

El campo *urn* corresponde con la identificación del sensor del que se ha solicitado la información. Los campos *phenomenon* y *value* indican el fenómeno físico que se está midiendo y la magnitud de dicho fenómeno, respectivamente. En este caso en el que se solicita el valor de la temperatura del sensor 3243 se obtiene que es de 25.73. El campo *uom* indica la unidad de medida de dicho fenómeno, por lo tanto, ese valor que se proporciona en la respuesta de 27.73 está medido en grados Celsius.

Además, también se proporcionan dos datos importantes que han sido empleados para el desarrollo de la skill de este trabajo, que son la posición del sensor y la última vez que fue actualizado.

Otro punto de captura de datos de información es la plataforma OpenData de Smart Santander [24], cuya dirección es *http://datos.santander.es*. En esta plataforma se encuentran conjuntos de datos de diferentes categorías, como pueden ser tráfico, transporte, urbanismo o comercios. Además, esta plataforma alberga un repositorio de aplicaciones desarrolladas en base a estos datos.

5.3.2 Acceso a datos de FIWARE

La forma de acceder a los datos a través de FIWARE es muy similar a la del caso anterior. El modelo de datos para los proyectos de FIWARE se define mediante programación utilizando un esquema JSON, que incluye una serie de atributos, metadatos y propiedades para el envío de variables a través de código. En la Figura 5.19 se puede ver un ejemplo de respuesta de FIWARE.

Para acceder a la información de los sensores de la ciudad de Santander por medio de esta plataforma se dispone de una dirección concedida a través del departamento de la universidad. Para acceder a la información de un sensor realizo una petición HTTP a esa dirección seguido del siguiente *path*:

/v2/entities/urn:ngsi-ld:WeatherObserved:santander:environment:fixed:${numero}?options=keyValues&attrs=${phenomenon},dateObserved,location

En este *path*, donde está colocado ${numero} se coloca el número del sensor a consultar, y en ${phenomenon} el fenómeno atmosférico deseado. Esto será analizado más adelante, en el capítulo 4.

Por su parte, esta consulta devuelve un archivo en formato JSON con la información correspondiente.

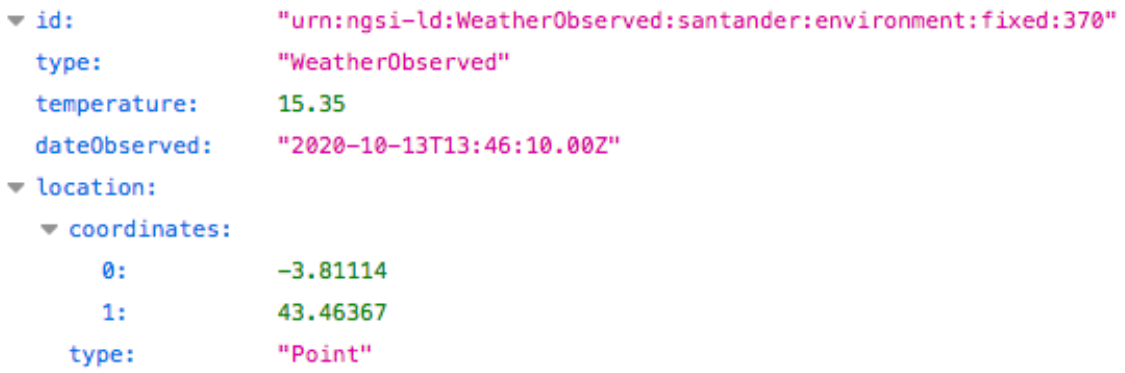


Figura 5.19: Respuesta de FIWARE

Se puede observar como en este respuesta devuelta por el sistema de FIWARE podemos obtener la información solicitada, habiendo elegido el sensor número 3248, y habiendo solicitado el fenómeno físico de la temperatura. Además de este valor de temperatura que se ha pedido, también se proporciona otra información relevante, la cual se empleará en la skill creada para este proyecto. Esa información es la posición del sensor, dada por sus coordenadas geográficas, y el valor del campo *dateObserved*. Este valor corresponde con la última vez que el sensor ha actualizado su información, muy importante para poder saber si se trata de un sensor que sigue en funcionamiento y que la información obtenida corresponde con valores actuales.

6 Reflexiones finales

Para finalizar, van a darse una serie de reflexiones respecto a este Trabajo de Fin de Grado. Lo primero va a ser un análisis sobre el transcurso de este, teniendo en el punto de mira los objetivos planteados y como se ha ido desarrollando el proceso de creación para el cumplimiento de estos objetivos. A continuación, va a ser expuesta la opinión que ha creado en mi persona el estudio de estas tecnologías y como percibo su futuro y las capacidades que tienen por desarrollar. Por último, quiero expresar que el final de este trabajo no significa el final del proyecto. Lo realizado aquí puede seguir adelante y no tiene que finalizar con la conclusión de este Trabajo de Fin de Grado.

6.1 Evolución y balance del trabajo

A lo largo del desarrollo del trabajo han sido numerosas las dificultades que han ido apareciendo, especialmente a nivel de realización. La situación actual con el COVÍD-19 ha provocado que el 100% del trabajo haya sido realizado suponiendo un gran esfuerzo, tanto para mí como para mi tutor.

Una vez fijado el proyecto, los problemas que se fueron presentando fueron de tipo técnico. Fue necesario cierto tiempo para asimilar el funcionamiento completo de una skill y de cómo es el desarrollo de una, especialmente en lo que a programación se refiere. Anteriormente no había tenido contacto con JavaScript ni con la creación de servicios REST, por lo que la mayor parte del tiempo se ha dedicado a resolver problemas de ese tipo. Las circunstancias le han dado un añadido de dificultad técnica, por el problema de la comunicación para resolver dudas y errores.

Aún así, se puede calificar el resultado de exitoso, habiendo obtenido una herramienta para mejorar el acceso de los usuarios a una plataforma de Ciudad Inteligente mediante el uso de un asistente virtual.

6.2 Conclusiones

Si las grandes empresas comentadas a lo largo de este escrito han apostado por la tecnología de los asistentes virtuales debemos plantearnos que por algún motivo será. Desde mi punto de vista creo que es una tecnología que se puede explotar mucho más allá de lo que se puede ver en el mercado hoy en día, llegando a crear producto y sistemas propios de películas de ciencia ficción.

Los asistentes virtuales tienen frente a ellos un mundo lleno de posibilidades para mejorar, como es el caso del gran empuje que tiene ahora mismo el Big Data. Este manejo de grandes cantidades de información y de información muy compleja puede resultar muy beneficioso para el entrenamiento de estos asistentes virtuales, a través del Machine Learning, llegando a ser unos sistemas altamente inteligentes

y convirtiéndose en el interfaz de contacto con el usuario en todos los dispositivos que nos podamos imaginar.

Esta sustitución de los dispositivos tradicionales que permiten que el usuario interactúe mediante botones o pantallas táctiles está a la orden del día debida a la crisis sanitaria del COVID-19. Una interfaz sin necesidad de tocar ningún botón ni pantalla permitiría evitar contactos en numerosos espacios públicos. Por poner algunos ejemplos se me ocurren los siguientes:

- En el metro, a la hora de recargar el abono de transporte, es necesario interactuar con una pantalla para seleccionar dicha opción. Esta pantalla táctil podría ser sustituida por un asistente virtual que te pregunte que es lo que deseas hacer, y que además de ofrecer la respuesta por audio se puedan seguir usando las pantallas para mejorar la usabilidad.

- Un ejemplo similar al anterior es el que tienen algunos cadenas de comida rápida en sus establecimientos. Estos lugares disponen de una serie de pantallas táctiles para realizar los pedidos. Como en el caso anterior se podrían sustituir por asistentes virtuales a los que el usuario le pide lo que desea.

Una vez destacada la importancia que creo que van a tener los sistemas de control de voz en el futuro, debo decir que pienso que nada de esto puede ser como se espera. No sería la primera vez que una gran tecnología que parece ser una revolución se queda en nada. Se me ocurre el ejemplo del 3D, que estuvo integrado hace unos años en gran cantidad de televisores, y hoy en día ya no se ve, siendo sustituido por las televisiones 4K o las nuevas 8K.

Con los sistemas IoT y el modelo de Smart City no creo que pase eso. Esto ha venido para quedarse y es fundamental desarrollar sistemas que favorezcan el crecimiento de este sector ara poder asegurar una gestión eficiente de los recursos de la ciudad y poder crear una ciudad ecológica y sostenible. Acercar todo esto al usuario para concienciar de su importancia y mejorar la usabilidad creando sistemas amigables es el objetivo de desarrolladores y demás profesionales del mundo TIC, y este proyecto es mi forma de contribuir a esto mismo.

6.3 Líneas de investigación futuras

Como ya he comentado anteriormente no creo que este proyecto llegue a su fin pese haber concluido este trabajo. Siempre hay mejoras que se pueden introducir o nuevas herramientas similares en la línea de la que aquí se ha presentado. De esta skill se me ocurren algunas mejoras como poder solicitar la información de un sensor preguntando por una calle, en lugar de por el número del sensor.

Creo que este proyecto puede ser la base para proyectos futuros de otros desarrolladores, que, a parte de mejorarla, podrían llegar a la creación de otras que permitan, por ejemplo, conocer el estado del tráfico en un lugar para elegir la ruta óptima mientras nos encontramos conduciendo, reduciendo la distracción.

Otra idea que surgió a raíz del tema del COVID-19 es poder conocer el aforo en espacios públicos mediante la instalación de sistemas de detección de rostros sin reconocimiento de identidad. De este modo habría un sistema que llevase el control de gente que se encuentra en un determinado espacio, y si el usuario tiene intención de ir a dicho lugar puede consultar al asistente virtual cuál es el porcentaje de ocupación de dicho lugar. Esta plataforma no tendría ese reconocimiento de identidad para respetar la privacidad de las personas allí presentes.

Son numerosas las ideas que surgen y en las que se puede explotar esta tecnología, y esa es la razón por la que le auguro un gran futuro.

Bibliografía

[1] Asociación para la Investigación de Medios de Comunicación, «AIMC Blog,» 17 octubre 2019. [En línea]. Available: https://www.aimc.es/blog/altavoz-inteligente-continua-carrera-conquistar-hogar-espanol/.

[2] C. d. l. Cruz, «BlogThinkbig,» 25 abril 2019. [En línea]. Available: https://blogthinkbig.com/asistentes-virtuales-usuarios.

[3] C. del Castillo, «elDiario,» 21 septiembre 2020. [En línea]. Available: https://www.eldiario.es/tecnologia/proyecto-calo-desconocido-padre-militar-siri-alexa-cortana-resto-asistentes-virtuales_1_6227243.html.

[4] J. Carbonell, «Centro Virtual Cervantes,» [En línea]. Available: https://cvc.cervantes.es/obref/congresos/sevilla/tecnologias/ponenc_carbonell.htm.

[5] «TICBeat,» 4 octubre 2020. [En línea]. Available: https://www.ticbeat.com/innovacion/pnl-como-aprenden-las-maquinas-a-hablar-como-los-humanos/.

[6] P. G. Bejerano, «elDiario,» 1 diciembre 2013. [En línea]. Available: https://www.eldiario.es/autores/pablo_g_bejerano/.

[7] A. Gomez Sanchez y F. J. Perez Sabroso, 2014. [En línea]. Available: https://www.it.uc3m.es/jvillena/irc/practicas/13-14/02.pdf.

[8] F. C. Nolla, «Centro Virtual Cervantes,» [En línea]. Available: https://cvc.cervantes.es/obref/congresos/sevilla/tecnologias/mesaredon_casacuberta.htm.

[9] Apple, «Siri Kit Documentation,» [En línea]. Available: https://developer.apple.com/documentation/sirikit.

[10] Google, «Google Assistant Documentation,» [En línea]. Available: https://developers.google.com/assistant/.

[11] R. Fernández, «Hipertextual,» 8 octubre 2020. [En línea]. Available: https://hipertextual.com/2020/10/asistente-google-contara-con-nueva-funcion-invitados.

[12] Microsoft, «Cortana Skills Documentation,» [En línea]. Available: https://docs.microsoft.com/en-us/cortana/skills/.

[13] «Alexa Skills Kit Documentation,» [En línea]. Available: https://developer.amazon.com.

[14] Enerlis, Ernst and Young, Ferrovial y Madrid Network, Libro Blanco Smart Cities, Madrid, 2012.

[15] «Energías renovables,» 10 julio 2019. [En línea]. Available: https://www.energias-renovables.com/ahorro/los-vehiculos-producen-el-13-de-la-20190710.

[16] Comisión de Ciudades Digitales y del Conocimiento de CGLU, «Smart Cities Study 2017: Estudio Internacional sobre la situación y el desarrollo de las TIC, la innovación y el conocimiento de las ciudades,» Bilbao, 2017.

[17] Cámara Valencia, «TIC Negocios - Cámara Valencia,» [En línea]. Available: https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-14-dispositivos-i-internet-de-las-cosas-iot/#Sensores_y_actuadores.

[18] «Smart Santander Documentation,» [En línea]. Available: https://api.smartsantander.eu/docs.

[19] «FIWARE,» [En línea]. Available: https://www.fiware.org/.

[20] Amazon, «AWS Lambda,» [En línea]. Available: https://docs.aws.amazon.com/lambda/index.html.

[21] «Ngrok,» [En línea]. Available: https://ngrok.com/.

[22] «Express Web Framework,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs.

[23] «MomentJS,» [En línea]. Available: https://momentjs.com/.

[24] Ayuntamiento de Santander, «Santander Datos Abiertos,» 2013. [En línea]. Available: http://datos.santander.es/.

[25] P. Berrone y J. E. Ricart, «Índice IESE Cities in Motion 2018,» 2018.

[26] L. Ohannesian, «Alexa Blogs,» 24 enero 2019. [En línea]. Available: https://www.developer.amazon.com/es/blogs/alexa/post/afa8c3b3-945c-4721-8363-44394ed4729c/alexa-hosted-skills-beta-is-now-available-to-all-developers.

[27] «AWS Documentation,» [En línea]. Available: https://aws.amazon.com/es/.

[28] G. Viscuso, «De Cero a Héroe: Construyendo Skills Alexa,» 13 marzo 2020. [En línea]. Available: https://www.youtube.com/playlist?list=PL2KJmkHeYQTNzra-T2ayhV_84dHYCShAQ.

[29] L. Fischer, «Guía para principiantes de HTTP y REST,» 4 diciembre 2016. [En línea]. Available: https://code.tutsplus.com/es/tutorials/a-beginners-guide-to-http-and-rest--net-16340.